НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»
Завідувач кафедри _____
А.І.Петренко
(підпис) (ініціали, прізвище)
"___" _____2016 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код та назва спеціальності)

на тему: «Автоматизовані засоби парсингу проектів дистанційного навчання в Cloud для заповнення сайту http://dl-cloud.kpi.ua»

Виконав: студент IV курсу, групи групи ДА-22
(шифр групи)

Ханенко Олександр Андрійович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник к.т.н., доц. Цурін О. П. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент _____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст.. викладач Бритов О.А. _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань. Студент

_____
(підпис)

Київ – 2016 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування

7.050103, 8.050103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

«___»_____2016 р.

ЗАВДАННЯ на дипломний проект (роботу) студенту

Ханенку Олександру Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) «Автоматизовані засоби парсингу проектів дистанційного навчання в Cloud для заповнення сайту http://dl-cloud.kpi.ua»
керівник проекту (роботи) Цурін О. П., к.т.н., доц.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від 12 травня 2016 р. № 50-ст

2. Строк подання студентом проекту (роботи) 08.06.2016

3. Вихідні дані до проекту (роботи)
Дослідити алгоритми парсингу даних в Cloud. Для реалізації використовувати спеціальні бібліотеки та фреймворки Python. Вхідними даними є веб-ресурси Cloud. Можливість вибору алгоритму оптимізації. Передбачити можливість оновлення контенту на веб-ресурсах. Результати роботи програми мають

включати наступні пункти: парсинг контенту із заданих веб-ресурсів, розміщення оброблених даних на http://dl-cloud.kpi.ua.

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)
    1. Проаналізувати подібні рішення.
    2. Розглянути алгоритми оптимізації та обрати варіант для розробки.
    3. Розробити алгоритм роботи.
    4. Розробити архітектуру системи.
    5. Розробити програмну систему та протестувати її.
    6. Проаналізувати результати роботи програми, зробити висновки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)
    1. Блок-схема алгоритму парсингу план-графіку
    2. Архітектура системи – плакат.
    3. Результати роботи програми - плакат.
6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Семенченко Н.В., професор		

7. Дата видачі завдання 01.02.2016

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2016	
2	Збір інформації	15.02.2016	
3	Вивчення алгоритмів парсингу веб-сайтів	28.02.2016	
4	Розробка алгоритму та структури системи	10.03.2016	
5	Розробка плану тестування	15.03.2016	
6	Розробка програмної моделі	25.03.2016	
7	Розробка опису	25.04.2016	
8	Тестування додатку та отримання результатів	30.04.2016	
9	Оформлення дипломної роботи	31.05.2016	
10	Отримання допуску до захисту та подача роботи в ДЕК	08.06.2016	

Студент _____ О. А. Ханенко
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ О. П. Цурін
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

До бакалаврської дипломної роботи Ханенка Олександра Андрійовича

на тему: “Автоматизовані засоби парсингу проектів дистанційного навчання в Cloud для заповнення сайту http://dl-cloud.kpi.ua”

Дана дипломна робота присвячена дослідженню технологій парсингу різних за структурою веб-сайтів, виявлення їх недоліків, виявлення концепції рішень та вибору найбільш правильного алгоритму парсингу, враховуючи структуру конкретних веб-ресурсів.

Метою роботи є створення сервісу парсингу веб-сайтів з науковим контентом, розміщення отриманих даних на сайті http://dl-cloud.kpi.ua, таким чином це зацікавить цільову аудиторію тим, що знайти потрібну інформацію можна буде на одному сайті та перейшовши по посиланню, користувач отримає сторінку з корисним для нього матеріалом.

В процесі розробки досліджувалися методи та засоби парсингу різних за структурою веб-сайтів, проведена їх порівняльна характеристика. Також було розглянуто методи парсингу сторінок з динамічним контентом.

Загальний об’єм роботи:

Ключові слова: парсинг, веб-сайт, посилання, контент, інформація, алгоритм, структура, веб-ресурс.

АННОТАЦИЯ

К бакалаврской дипломной работы Ханенка Александра Андреевича

на тему: "Автоматизированные средства парсинга проектов дистанцион- ного обучения в Cloud для заполнения сайта http://dl-cloud.kpi.ua"

Данная дипломная работа посвящена исследованию технологий парсинга различных по структуре веб-сайтов, выявление их недостатков, выявление концепции решений и выбора наиболее правильного алгоритма парсинга, учитывая структуру конкретных веб-ресурсов.

Целью работы является создание сервиса парсинга сайтов с научным контентом, размещение полученных данных на сайте http://dl-cloud.kpi.ua, таким образом это заинтересует целевую аудиторию тем, что найти нужную информацию можно будет на одном сайте и перейдя по ссылке, пользователь получит страницу с полезным для него материалом.

В процессе разработки исследовались методы и средства парсинга различных по структуре веб-сайтов, проведена их сравнительная характеристика. Также были рассмотрены методы парсинга страниц с динамическим контентом.

Общий объем работы:

Ключевые слова: парсинг, веб-сайт, ссылка, контент, информация, алгоритм, структура, веб-ресурс.

ANNOTATION

To bachelor thesis work of Oleksandr Khanenko
entitled “Automated parsing means of distance learning projects in the Cloud to fill site http://dl-cloud.kpi.ua”

This thesis is devoted to researching technologies parsing various websites structure, identifying their weaknesses, identifying solutions and concepts to select the most right parsing algorithm, considering the structure of the specific web resources.

The purpose is the creation of the service of parsing websites with scientific content, placing received data on http://dl-cloud.kpi.ua site, so it will interest the target audience in order to find the desired information can be on the same site and clicking on the link, the user will get a page with useful material for him.

During the development process were explored ways and means of parsing various websites structure, carried out their comparative characteristics. Also, were examined methods of parsing pages with dynamic content.

Total volume of work:

Keywords: parsing, website, link, content, information, algorithm, structure, web resource.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Web – система доступу до пов'язаних між собою документів на різних комп'ютерах, підключених до Інтернету

HTML – HyperText Markup Language

URL – Uniform Resource Locator

HTTP – HyperText Transfer Protocol

XML – eXtensible Markup Language (розширювана мова розмітки)

БД – база даних

NoSQL – not only Structured query language

CSV - Comma-Separated Values

JSON - JavaScript Object Notation

DOM - Document Object Model

URL - Uniform Resource Locator

CAPTCHA - Completely Automated Public Turing test to tell Computers and Humans Apart

DFD - Data Flow Diagrams

ВСТУП

Одною із основних потреб людського життя є інформація. Ми з кожнем днем взнаєм щось нове, читаєм, дивимось ТБ, слухаємо радіо, спілкуємося з людьми. Інколи людина, користуючись Інтернетом, втрачає багато свого дорогоцінного часу, щоб найти конкретну інформацію, адже потрібно переглянути не один сайт, наткнутися на рекламу, інтерфейс деяких сайтів є доволі таки складним і найти там щось потрібне займає багато часу. Інколи людина заходить в тупік, потрібна інформація не знайдена, залишається обмаль часу на пошук. Виникає потреба в пришвидшені процесу збору інформації по сайтах.

В даний час всі процеси, де застосовується синтаксичний аналіз, використовують парсери - програми для проведення візуального або програмно-автоматизованого синтаксичного і лексичного аналізу або розбору будь-якого документа з метою вилучення з нього необхідних даних. Це і різні автоматизовані перекладачі з однієї мови на іншу, і транслятори мов програмування, які формують програмний код на машинно-орієнтовану мову, це і мова SQL-запитів і тому подібні застосування.

Перш, ніж розробляти парсер, необхідно визначитися з його визначенням і призначенням.

Отже, парсер контенту - це не що інше, як скрипт, здатний сортувати інформацію, виділяючи найважливішу і обробляючи її згідно з алгоритмом, створеному для вирішення того чи іншого завдання. Для чого, наприклад, можна використовувати парсер контенту.

По-перше, як відомо, найкращі сайти - це ті Інтернет-ресурси, на яких є цікава актуальна інформація. Нікому не потрібні вчорашні новини і сенсації, наприклад. Також, коли мова йде про сайти-обмінники валют, де необхідно змінювати інформацію про курс валют часом по кілька разів на день, створення парсера вкрай необхідно. Скрипт в таких випадках буде виконувати всю роботу сам, цілодобово відстежуючи зміни курсу валют в Нацбанку.

По-друге, парсер необхідний для автоматичного оновлення вашого Інтернет-ресурсу. Користувачі, які зайшли на вашу сторінку один раз і виявили там застарілу інформацію, більше ніколи не повернуться на неї. Саме тому для збереження постійних користувачів, а також для залучення нових необхідне регулярне оновлення інформації на вашій інтернет-сторінці.

По-третє, парсер контенту - ідеальний інструмент для миттєвого наповнення сайту корисними даними. Так, коли в мережі величезна кількість сайтів різної спрямованості, лише мала їх частина виявляється в кінцевому підсумку корисними користувачам. Саме тому, важливо, щоб інформація на вашій веб-сторінці була не тільки актуальною, але ще і корисною.

І, по-четверте, - централізація даних. Відомо, що інформації в мережі предостатньо, при цьому самої різної. Вся проблема в тому, що вся ця інформація розкидана по безлічі Інтернет-рессурсах і зібрати її не так просто. Використовуючи парсер контенту, можна об'єднати вичерпну кількість корисної інформації у себе на сайті, тим самим залучаючи все більше відвідувачів. Такий хід - відмінний варіант для далекоглядних розробників, які піклуються про те, щоб навіть випадкові відвідувачі, ставали постійними. Саме для вирішення такої задачі присвячена ця дипломна робота.

Ми розробимо програму-парсер, яка буде збирати дані з провідних проектів дистанційного навчання. Отримані дані в подальшому будуть розміщені на сайті http://dl-cloud.kpi.ua, що дасть змогу студентам відвідати цей ресурс і найти потрібні уроки по тій чи іншій дисципліні, не витрачаючи на це велику кількість часу. Потрібно всього лиш зайти на наш сайт, вибрати потрібну категорію, і в результаті користувач получить список потрібних йому матеріалів, перейшовши по посиланню, студент приступить до вивчення конкретних курсів. Не потрібно більше заморочуватись з пошуком інформації, переходити по необмеженій кількості посилань, задавати фільтрацію даних, натикатись на кучу реклами, все це за вас зробить автоматизована програма-парсер.

Вище перераховані основні переваги парсеру контенту, завдяки яким стає ясно, що даний скрипт можна використовувати тільки на користь при грамотному підході і бажанні розробити сайт з високим ступенем відвідуваності.

1. ВІДОМІ ТЕХНОЛОГІЇ ТА АЛГОРИТМИ ПАРСИНГУ ВЕБ-САЙТІВ

Парсинг - це лінійне зіставлення послідовності слів з правилами мови. Поняття «мова» розглядається в найширшому контексті. Це може бути звичайна людська мова (наприклад, українською), яка використовується для комунікації людей. А може і формалізована мова, зокрема, будь-яка мова програмування.

Парсинг сайтів - послідовний синтаксичний аналіз інформації, розміщеної на Інтернет-сторінках. Текст Інтернет-сторінок – це ієрархічний набір даних, структурований за допомогою людських і комп'ютерних мов. Людською мовою надана інформація, знання, заради яких, власне, люди і користуються Інтернетом, взагальному контент. Комп'ютерні мови (HTML, JavaScript, CSS) визначають як інформація виглядає на моніторі, розмітку сторінки, її стиль.

1.1 Актуальність задачі

Застосування парсинга може бути актуальним в наступних випадках:

- Якщо необхідно автоматично оновлювати сторінки вашого сайту, тобто автоматично додавати статті, новини або інший контент;
- Для підтримки актуальності використовуваної на сайті інформації;
- Якщо необхідно об'єднати інформацію, так як вона часто перебуває на різних сайтах.
- Великі обсяги. В епоху бурхливого зростання мережі і жорстокої конкуренції вже всім ясно, що успішний веб-проект немислимий без розміщення великої кількості інформації на сайті. Сучасні темпи життя призводять до того, що контенту має бути не просто багато, а

дуже багато, в кількостях, що набагато перевищують межі, можливі при ручному заповненні.

- Часте оновлення. Обслуговування величезного потоку динамічно мінливої інформації не в силах забезпечити одна людина або навіть злагоджена команда операторів. Часом інформація змінюється щохвилини і в ручному режимі оновлювати її навряд чи доцільно.

В процесі парсинга застосовуються скриптові мови програмування: PHP, Perl, Ruby, Python, JavaScript і багато інших. Він здійснюється шляхом написання так званого парсеру. Парсер - це програма, за допомогою якої відбувається автоматична обробка сторінок сайту для отримання необхідних даних. Парсер сайтів створюється для збору великої бази контенту на сайт і для здійснення пошуку необхідної для користувачів інформації.

Останнім часом Інтернет став все більше поповнюватися новими інтернет - магазинами. Тому що на сьогоднішній день наявність власного інтернет - магазину є зручним і перспективним напрямком в бізнесі. Ви можете створити його і самостійно, але якщо ви не дуже розбираєтеся в програмуванні, то можна створити інтернет - магазин за допомогою парсинга. А наповнити ваш сайт потрібною інформацією допоможе парсер інтернет-магазинів. Він здатний парсити товари з необхідної вам категорії. Використовуючи парсер, можна самостійно створювати будь-які магазини.

Парсинг сайтів є ефективним рішенням для автоматизації збору і зміни інформації.

У порівнянні з людиною, комп'ютерна програма-парсер:

1. швидко обійде тисячі веб-сторінок;
2. акуратно відокремить технічну інформацію від «людської»;
3. безпомилково відбере потрібне і відкине зайве;
4. ефективно упакує кінцеві дані в необхідному вигляді.

Результат (будь то база даних чи електронна таблиця), звичайно ж, потребує подальшої обробки. Втім, подальші маніпуляції із зібраною інформацією вже до теми парсинга не належать.

1.2 Інструменти для написання парсера

Написання парсерів не вимагає монументальних знань про використовувану мову програмування будь то PHP, Ruby або Python. Також необов'язково мати академічні відомості про супутні технології. Однак, дещо доведеться вивчити. Перерахуємо веб-технології, які доведеться знати кожному, кого цікавить професійне створення синтаксичних аналізаторів:

- Щоб створити первинний алгоритм роботи майбутнього парсеру, доведеться проаналізувати вихідний код сторінок сайту-донора. Само собою, без знань (хоча б на середньому рівні) HTML, CSS і JavaScript ніяк не обійтися.
- Для більш глибокого занурення в тему рекомендується до вивчення технологія DOM, що дозволяє з максимальним ефектом працювати з ієрархічним деревом веб-документа.
- На найважливішому і складному етапі - написанні аналізатора - потрібно знання будь-якого з інструментів текстової обробки. Один з варіантів для пошуку потрібних шматків тексту скористатися регулярними виразами. Безумовно, «регулярки» користуються заслуженою славою як могутнього засобу для вирішення багатьох нетривіальних завдань. Однак цей спосіб не є найкращим, а часто і небажаним. По-перше, все-таки, регулярні вирази досить складно освоїти для подальшої роботи. Для цього потрібно володіти досить специфічним мисленням. По-друге, html-код на більшості сайтів не валідний, а часто і некоректний. Навіть найзапекліші фахівці можуть заплутатися в метасимволах і квантифікаторах, намагаючись передбачити всі випадки даної задачі. Тому, оптимальним виходом буде не винахід колеса, а використання готових бібліотек для парсинга, трохи нижче вказані найпопулярніші рішення для кожного з мов. Втім, це зовсім не означає що регулярні вирази можна і зовсім не

вивчати. Найчастіше саме за допомогою них зручно вирішувати багато проблем, які не взмозі вирішити конкретна бібліотека для парсинга.

- Для ефективної роботи з ієрархічними структурами даних - бажано володіти парадигмою об'єктно-орієнтованого програмування. Семантичне дерево можна будувати і за допомогою багатовимірних асоціативних масивів, але, цей спосіб не логічний у випадку, коли дерево містить велику кількість вузлів, ООП відмінно підтримується всіма мовами розглянутими на сторінках цього сайту.
- Фінальна обробка результатів передбачає збереження даних в структурованому вигляді. Зазвичай на виході потрібна база даних. Так що знадобляться міцні знання SQL, і, як мінімум, MySql і PostgreSQL.
- Не обійтися без впевненого володіння функціями для роботи з файлами. Цілком можливо, дані доведеться дописувати в CSV-файли або конвертувати в електронні таблиці.
- Відомості про XML і XPath можуть знадобитися як на етапі синтаксичного аналізу (іноді доводиться парсити не сайти, а RSS-потоки), так і на стадії кінцевого збереження результатів (тобто, на основі сторонніх сайтів іноді необхідно створити все ту ж стрічку RSS).
- Іноді спарсені дані заливаються в нову базу даних за допомогою JSON. Дану javascript-технологію теж необхідно освоїти, нічого складного вона з себе не представляє.

1.3 Етапи парсинга

Парсинг html-сторінки вдає із себе процес, який можна розбити на три етапи:

1. Отримання початкового коду веб-сторінки - скачати програмний код тієї сторінки сайту, з якої необхідно витягти інформацію. У різних мовах для цього передбачені різні способи, наприклад, в PHP

найчастіше використовують бібліотеку cURL або ж вбудовану функцію file_get_contents.

2. Витяг з html-коду необхідних даних. Отримавши сторінку, необхідно обробити її - відокремити звичайний текст від гіпертекстової розмітки, вистроїти ієрархічне дерево елементів документа, коректно зреагувати на невалідний код, виокремити зі сторінки саме ту інформацію, заради якої і відбувається весь цей процес. Можна, звичайно ж, використовувати для цього регулярні вирази, проте є більш зручний шлях - спеціалізовані бібліотеки.
3. Фіксація результату. Благополучно обробивши дані на сторінці, потрібно їх зберегти в необхідному вигляді для подальшої обробки. Спарсені дані зазвичай заносяться в базу даних, однак є й інші варіанти. Іноді потрібно записати в CSV-файл або будувати ієрархічні JSON-структури, іноді конвертувати в excel-таблицю, а може навіть згенерувати динамічний rss-потік.

Як правило, потрібно спарсити не одну сторінку сайту-донора, а безліч, може навіть і все. У цьому випадку після проходження кроків 1-3 у алгоритм парсера повинен бути закладений перехід на наступну сторінку сайту, щоб і з неї вилучити весь необхідний матеріал.

Обхід всіх потрібних сторінок сайту забезпечується різними способами.

- По-перше, обробляючи чергову сторінку, парсер можна навчити не тільки отримувати необхідні дані, але і заносити в свою базу даних всі внутрішні посилання, що зустрічаються на шляху. Звертаючись до свого сховища лінків, програма послідовно відвідує сторінки сайту, до тих пір поки не обійде їх всіх.
- По-друге, при первинному аналізі сайту найчастіше можна простежити логіку формування url для сторінок. І потім, генерувати адреси відповідно до виявлених закономірностей.
- По-третє, деякі парсери розраховані, як не дивно, на «ручний» обхід веб-ресурсу. Користувач, клікаючи по посиланнях, сам вирішує які

сторінки відвідувати, які ні. А програма у фоновому режимі запам'ятовує необхідні дані.

Зрозуміло, поєднувати різні методи ніщо не заважає.

В цілому парсинг сайтів складається з приблизно наступних етапів:

- Аналіз сайту: визначення структури сайту і шаблону даних і, на цьому етапі корисним буває вивчити файл robots.txt, xml карту сайту, пошук по сайту, видачу пошуковиків для сайту.
- Підготовка виразів (xpath або css селектори) для отримання необхідних даних зі сторінок.

1.4 Імпорт контента

Перш ніж аналізувати будь-який html-код - його треба звідкись взяти. Очевидність постановки даної задачі зовсім не означає простоту реалізації. У більшості випадків рішення зводиться до завантаження інтернет-сторінки стандартними методами, проте не завжди все виявляється так просто.

Деякі сайти з легкістю розпізнають ботів для парсинга (як би ті майстерно не маскувалися під справжніх користувачів) і блокують отримання сторінок. Переважно непросто дістатися до вмісту фреймів і ділянок, підвантажуваних через Ajax. Дуже часто для отримання потрібних сторінок необхідна авторизація, виникає проблема у вигляді сесій і coockie. Окремо потрібно відмітити випадки, коли веб-сторінка повністю генерується на стороні клієнта, динамічно формуючись за допомогою JavaScript. Не рідкість ситуація, в якій іноземні сайти-донори ігнорують будь-які запити по IP з країн СНД. Зрештою, доводиться працювати з веб-сайтами, які повільно чи частково завантажуються або взагалі працюють через раз.

Іноді отримання інформації є набагато більш складним завданням ніж наступний синтаксичний розбір. Детально як з цим боротись розглянемо пізніше.

1.5 Синтаксичний аналіз

Отже, отримавши вихідний код сторінки, можна починати її аналіз, відокремлюючи ту інформацію, заради якої парсився сайт.

Цілком резонне питання - який інструмент вибрати для обробки? У минулі часи у програмістів не було особливого вибору крім як вдаватися до аналізу сторінки за допомогою регулярних виразів. Але, враховуючи їхню складність і опираючись на те, що на даний момент є набагато простіші шляхи вирішення даного питання, детально розглядати і реалізовувати цей метод не будемо.

Неприємною реальністю в використанні регулярних виразів є напів-коректний html-код більшості сайтів. Хоча веб-браузери в більшості випадків відображають все вірно, небагато веб-ресурсів притримуються 100% відповідності стандартам W3C.

В принципі, можна самостійно побудувати дерево документа і потім працювати з ним за допомогою технології Document Object Model. Після цього буде простіше аналізувати некоректний html-кодом. Однак, часу і сил на написання власного інтерпретатора сторінок піде чимало.

Згодом з'явилися чудові безкоштовні рішення, покликані полегшити життя кодерам. Написання парсерів спростилося завдяки спеціалізованим бібліотекам для парсинга. Більше немає потреби у різноманітних текстових масках - все вже давно зроблено за нас.

1.6 Експорт даних

Матеріал, отриманий з розпарсенного сайту, необхідно упакувати у вигляді, придатному для подальшого використання. Конкретний формат залежить від того як в подальшому буде оброблятися зібрана інформація.

Найчастіше це бази даних MySQL або PostgreSQL. Заливати в БД можна не тільки за допомогою запитів SQL, але і за допомогою JSON через Ajax. У багатьох випадках із спарсенного контенту за допомогою XML формується RSS-потік, що вельми зручно при використанні даних, без процедури рерайтинга. Іноді результат парсинга поміщають в CSV-файл - оскільки цей текстовий формат дуже простий у подальшій обробці, легко конвертується в SQL-запити і без проблем відкривається в Excel. У спеціальних випадках потрібно, щоб кінцеві дані були представлені у вигляді електронних таблиць XLS.

1.6.1 SQL

При парсингу кожної сторінки сайту-донора, витягнуті дані, як правило, тут же заносяться в базу даних. Для повної реалізації даної задачі нам не обійтися без впевнених знань про MySQL і PostgreSQL - двох найбільш використовуваних СУБД для веб-розробок.

Зазвичай у вигляді БД спарсенний контент і потрібний для перенесення на інший веб-ресурс. Однак часто наповненням бази даних справа не закінчується. Інформацію з SQL доводиться конвертувати в інші формати: JSON, текстовий CSV (який, в свою чергу, є зручним буфером для подальшої обробки), таблично-процесорний XLS (в PHP є спеціальні додаткові бібліотеки для перетворення SQL-даних в електронні таблиці Excel), XML (часом фінальним акордом парсерной сюїти є веб-серверний RSS-агрегатор).

Ну, і звичайно ж, доводиться іноді перетворювати даних з MySql в PostgreSQL і назад, бо бажана веб-програмістом і необхідна для конкретного завдання СУБД можуть не збігатися.

1.6.2 CSV

CSV (англ. Comma-Separated Values - значення через кому) - формат текстових файлів для зберігання табличної інформації. Один рядок - один запис

в таблиці. Роздільник значення полів у записі - кома. Втім, замість коми можна використовувати і інші символи.

Формат CSV популярний оскільки дуже простий і універсальний. Його можна легко:

- правити в будь-якому текстовому редакторі;
- відкривати в Microsoft Excel і OpenOffice.org Calc;
- змінювати функціями PHP для роботи з файлами;
- конвертувати в SQL і назад.

При парсингу досить-таки часто у Вас можуть запитати спарсені дані в цьому форматі. Що не викликає ні найменших труднощів і дуже зручно для подальшої обробки інформації.

Приклад CSV-файлу (рис. 1.1 – Структура CSV-файлу):

	A	B	C
1	First Name	Last Name	Email Address
2	Jane	Doe	jane@gmail.com
3	John	Doe	john@gmail.com
4	Chris	Perkins	cperk@gmail.com
5	Eva	Davis	eva@gmail.com
6	Mitchell	Tarver	mitch@gmail.com
7	Nathan	Woodward	nathanwoodward@gmail.com

Рисунок 1.1 - Структура CSV-файлу

1.6.3 RSS

XML-документи (як технологія для RSS) в парсингу використовується в двох протилежних задачах.

По суті справи RSS-потік - це спарсена інформація з потрібного нам сайту. Якщо канал є джерелом контенту, то завдання програміста зводиться до нескладної фільтрації або об'єднання декількох RSS-feed'ів з різних джерел в один загальний. По суті справи, парсер в даному випадку представляє з себе веб-серверний RSS-агрегатор, настроєний на вирішення конкретного завдання.

З іншого боку, спарсена інформація часто сама є основою для створення RSS-каналу. На основі отриманих даних формується стрічкове зведення,

інформуюче про наповнення сторонніх сайтів і періодичних змінах на їхніх сторінках.

1.6.4 XLS

Іноді спарсенний матеріал потрібно представити у вигляді електронної таблиці Excel. Безумовно, електронні таблиці неможливо редагувати як звичайний текстовий файл. Досить відкрити документ XLS в текстовому редакторі (Notepad ++, Sublime Text, наприклад) і виявити там набір кракозябри, щоб прийти до висновку про марність стандартних функцій для роботи з файлами. Просто дописати в кінець файлу чергові дані (як у випадку з CSV), витягнуті з сайту-донора, не вийде.

Що ж, якщо стандартні функції не підходять, то на зміну їм використовують нестандартні. Існує чимало бібліотек, покликаних полегшити роботу з книгами Excel. Підключивши їх, можна як зчитувати інформацію з готових електронних таблиць, так і засобами мов веб-програмування формувати XLS-документи. Додавати нові записи в таблицю можна безпосередньо відразу після вилучення контенту з сайту-донора. Або ж спарсити матеріал в базу даних SQL, а потім з БД сформувати електронну таблицю.

1.6.5 JSON

У міру обходу сторінок сайту-донора часто зібрана інформація тут же за допомогою запитів SQL заноситься в базу даних. Альтернативою є JSON - текстовий формат обміну даними оснований на JavaScript.

У даного методу чимало безперечних переваг перед тим же SQL. JSON відрізняють мовонезалежність, крайня простота і лаконічність синтаксису, наявність готових рішень для обробки даних. Але головна перевага - це можливість прямої взаємодії браузера і сервера, що дозволяє реалізовувати

алгоритми пов'язані з фоновим занесенням інформації в базу даних, роботою з файловим кешем, серіалізацією, десеріалізацією динамічних структур та ін.

1.6.6 Document Object Model

DOM (Document Object Model) - багатоплатформовий багатомовний програмний інтерфейс, що надає додаткам і скриптам доступ (в тому числі і на зміну) до структури та змісту HTML, XHTML, XML-документів.

Document Object Model дозволяє ефективно вибудувати ієрархічне дерево веб-документа (рис. 1.2) для подальшої роботи з ним.

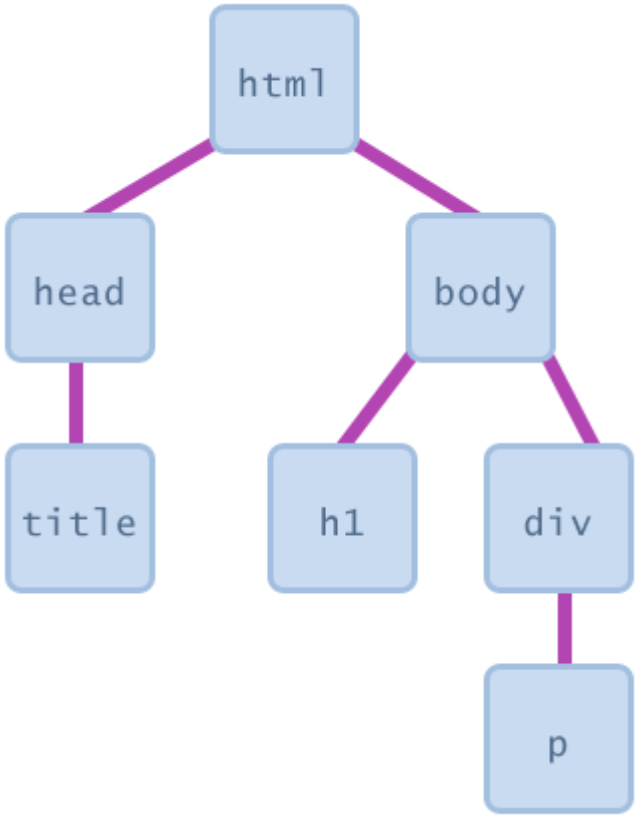


Рисунок 1.2 – Ієрархічне дерево веб-документа [3]

Велика частина бібліотек для парсинга основані на DOM.

DOM є нащадком Dynamic HTML Object Model - об'єктною моделлю веб-документів, розробленої для Netscape Navigator і Internet Explore 4 в 1997 році. Новинка дозволяла за допомогою JavaScript на стороні клієнта змінювати HTML-структуру документа і маніпулювати каскадних таблицями стилів.

Обидва браузера з часом перестали самостійно розвивати Dynamic HTML Object Model, яка зараз вважається морально застарілою технологією. Netscape згодом взагалі припинив існування, IE підтримує дану технологію з метою забезпечення сумісності.

Моделі, що існували до DOM1 і ввійшли в перший та наступні рівні, відомі як «DOM рівень 0» (DOM0, DOM level 0). Тут, зокрема, можна згадати document.images, document.forms, document.layers і document.all. Дані моделі не є специфікаціями DOM, назва «DOM рівень 0» це свідчення того, що передувало офіційним розробкам W3C.

До слова, розробники браузерів не зобов'язані інтерпретувати веб-документи з позиції Document Object Model. Але саме за допомогою DOM мова JavaScript отримує доступ до структури та змісту HTML-документів. Тому вибору - підтримувати чи ні - за великим рахунком, немає.

Поточною версією є DOM2, її взяли на озброєння всі браузерні движки. DOM3 поки що має експериментальний (рекомендований) статус і навіть основні браузери підтримують його не в повній мірі. В PHP5, Ruby1.9.x і Python3 також DOM3 реалізований лише частково.

1.7 Захист від парсингу та методи його обходу

1. Бан по IP адресу

Найпростішим і поширеним способом визначення спроб парсингу сайту є аналіз частоти і періодичності запитів до сервера. Якщо з якогось IP адресу запити йдуть занадто часто або їх занадто багато, то ця адреса блокується і щоб його розблокувати часто пропонується ввести каптчу.

Найголовніше в цьому способі захисту - знайти межу між природною частотою і кількістю запитів і спробами скрейпінга щоб не заблокувати ні в чому не винних користувачів. Зазвичай це визначається за допомогою аналізу поведінки нормальних користувачів сайту.

Прикладом використання цього методу може служити Google, який контролюється кількість запитів з певної адреси і видає відповідне попередження з блокуванням IP адреси і пропозицією ввести каптчу.

Є сервіси (на приклад distilnetworks.com), які дозволяють автоматизувати процес відстеження підозрілої активності на вашому сайті і навіть самі включають перевірку користувача за допомогою каптчі.

Обхід цього захисту здійснюється за допомогою використання декількох проксі-серверів, які приховують реальну IP-адресу парсера. Наприклад сервіси типу BestProxyAndVPN надають недорогі проксі, а сервіс SwitchProxy хоч і дорожче, але спеціально призначений для автоматичних парсерів і дозволяє витримати великі навантаження.

2. Використання облікових записів

У цьому способі захисту доступ до даних здійснюється тільки авторизованим користувачам. Це дозволяє легше контролювати поведінку користувачів і блокувати підозрілі акаунти незалежно від того, з якої IP адреси працює клієнт.

Прикладом може служити Facebook, активно контролює дії користувачів і блокує підозрілих.

Цей захист обходиться шляхом створення (в тому числі автоматичного) безлічі облікових записів (є навіть сервіси, які торгують готовими обліковими записами для відомих соціальних мереж, наприклад buyaccs.com і bulkaccounts.com). Істотним ускладненням автоматичного створення облікових записів може бути необхідність верифікації аккаунта за допомогою телефону з перевіркою його унікальності (так звані, PVA -Phone Verified Account). Але, в принципі, це теж обходиться шляхом покупки безлічі одноразових SIM-карт.

3. Використання CAPTCHA

Це теж поширений метод захисту даних від парсингу. Тут користувачеві для доступу до даних сайту пропонується ввести капчу (CAPTCHA). Істотним недоліком цього способу можна вважати незручність користувача в

необхідності введення капчі. Тому цей метод найкраще застосовувати в системах, де доступ до даних здійснюється окремими запитами і не дуже часто.

Прикладом використання каптчі для захисту від автоматичного створення запитів можуть служити сервіси перевірки позиції сайту в пошуковій видачі (наприклад http://smallseotools.com/keyword-position/).

Обходиться каптча за допомогою програм і сервісів по її розпізнаванню. Вони діляться на дві основні категорії: автоматичне розпізнавання без участі людини (OCR, наприклад програма GSA Captcha Breaker) і розпізнавання за допомогою людини (коли десь в Індії сидять люди і в режимі онлайн обробляють запити на розпізнавання картинок, наприклад може служити сервіс Bypass CAPTCHA). Людське розпізнання зазвичай більш ефективно, але оплата в даному випадку відбувається за кожну каптчу, а не один раз, як при покупці програми.

4. Використання складної JavaScript логіки

Тут в запиті до сервера браузер відсилає спеціальний код (або декілька кодів), які сформовані складною логікою написаною на JavaScript. При цьому, часто код цієї логіки обфусцированій і розміщений в одному або декількох підвантажуваних JavaScript-файлах.

Типовим прикладом використання даного методу захисту від парсинга є Facebook.

Обходиться це за допомогою використання для парсинга реальних браузерів (наприклад, за допомогою бібліотек Selenium або Mechanize). Але це дає цим методом додаткових переваг: виконуючи JavaScript, парсер буде проявляти себе в аналітиці відвідуваності сайту (наприклад Google Analytics), що дозволить вебмайстру відразу помітити недобре.

5. Динамічна зміна структури сторінки

Один з ефективних способів захисту від автоматичного парсинга - це часта зміна структури сторінки. Це може стосуватися не тільки зміна назв ідентифікаторів і класів, але навіть і ієрархії елементів. Це сильно ускладнює написання парсеру, але з іншого боку ускладнює і код самої системи.

З іншого боку, ці зміни можуть робитися в ручному режимі десь раз на місяць (або кілька місяців). Це теж істотно зіпсує життя парсерам.

Щоб обійти такий захист потрібне створення більш гнучкого і «розумного» парсеру або ж (якщо зміни робляться не часто) просто ручне виправлення парсеру, коли ці зміни відбулися.

6. Обмеження частоти запитів і обсягів даних при завантаженні

Це дозволяє зробити парсинг великої кількості даних дуже повільним і тому недоцільним. При цьому, обмеження необхідно вибирати виходячи з потреб типового користувача, що б не знизити загальну зручність користування сайтом.

Обходиться це за допомогою доступу до сайту з різних IP адрес або облікових записів (симуляція багатьох користувачів).

7. Відображення важливих даних у вигляді картинок

Даний спосіб захисту контенту дозволяє ускладнити автоматичний збір даних, при цьому зберігши візуальний доступ до них з боку звичайного користувача. Часто на картинки замінюються адреси електронної пошти та телефони, але деякі сайти примудряються замінювати картинками навіть випадкові літери в тексті. Хоча ніщо не заважає повністю виводити вміст сайту у вигляді графіки (будь то Flash або HTML 5), однак при цьому може істотно постраждати його індексованість пошуковими системами.

Мінус цього способу не тільки в тому, що не весь контент буде індексуватися пошуковими системами, але і в тому, що виключається можливість користувачеві скопіювати дані в буфер обміну.

Обходиться такий захист складно, швидше за все потрібно застосовувати автоматичне або ручне розпізнавання картинок, як і в разі капчи.

1.8 Висновки

У розділі було розглянуто основні особливості парсингу веб-сайтів, проблеми які виникають. Визначено актуальність задачі, роль програми-

парсера. Досліджено інструментарій за допомогою якого можна реалізувати дане завдання, а також проаналізовано варіанти синтаксичного розбору веб-сторінок, імпорту та експорту даних. Коротко ознайомлено зі способами захисту від парсингу та методами їх обходу.

2. АНАЛІЗ МОЖЛИВОСТЕЙ PYTHON ЯК ЗАСОБУ ДЛЯ СТВОРЕННЯ ВЕБ-ПАРСЕРА

Python - високорівнева мова програмування загального призначення, орієнтована на підвищення продуктивності розробника і читання коду. Синтаксис ядра Python мінімалістичний. У той же час стандартна бібліотека включає великий обсяг корисних функцій.

Python підтримує кілька парадигм програмування, в тому числі структурне, об'єктно-орієнтоване, функціональне, імперативне і аспектно-орієнтоване. Основні архітектурні риси - динамічна типізація, автоматичне керування пам'яттю, повна інтроспекція, механізм обробки виключень, підтримка багатопоточних обчислень і зручні високорівневі структури даних. Код в Python організовується у функції та класи, які можуть об'єднуватися в модулі (вони в свою чергу можуть бути об'єднані в пакети).

Еталонною реалізацією Python є інтерпретатор CPython, що підтримує більшість активно використовуваних платформ. Він поширюється під вільною ліцензією Python Software Foundation License, що дозволяє використовувати його без обмежень у будь-яких додатках, включаючи пропрієтарні. Є реалізації інтерпретаторів для JVM (з можливістю компіляції), MSIL (з можливістю компіляції), LLVM та інших. Проект PyPy пропонує реалізацію Python на самому Python, що зменшує витрати на зміни мови та постановку експериментів над новими можливостями.

Python - активно розвиваюча мова програмування, нові версії (з додаванням, зміною мовних властивостей) виходять приблизно раз в два з половиною роки. Внаслідок цього і деяких інших причин на Python відсутні стандарт ANSI, ISO або інші офіційні стандарти, їх роль виконує CPython.

Як наслідок, на цій мові створено немало зарекомендованих бібліотек, серед них і ті, які прекрасно справляються з парсингом веб-сторінок.

2.1 Завантаження веб-сторінок

Щоб сканувати веб-сторінки, нам спочатку потрібно, завантажити їх. Ось простий скрипт Python, який використовує urllib2 модуль Python, щоб завантажити URL:

```
import urllib2
def download(url):
      return urllib2.urlopen(url).read()
```

Коли передається URL, то ця функція буде завантажувати веб-сторінки і повертати HTML. Проблема з цим фрагментом виникає тоді, коли при завантаженні веб-сторінки, ми можемо зіткнутися з помилками, які знаходяться поза нашим контролем. Наприклад, задана сторінка, можливо, більше не існує. У цих випадках urllib2 згенерує заперечення і завершить виконання скрипта. Для того, щоб уникнути цього, більш надійною є версія, яка допоможе запобігти цим виняткам:

```
import urllib2
def download(url):
      print 'Downloading:', url
      try:
            html = urllib2.urlopen(url).read()
      except urllib2.URLError as e:
            print 'Download error:', e.reason
            html = None
      return html
```

Тепер, коли зустрінеться помилка завантаження, виняток перехоплюється і функція повертає None.

2.1.1 Повторна спроба завантаження

Часто помилки, що виникають при завантаженні носять тимчасовий характер, наприклад, веб-сервер перевантажений і повертає помилку 503 Unavailable Service. Для цих помилок, ми можемо повторити завантаження так, як проблема сервера може бути швидко вирішена. Проте, ми не хочемо, виконувати повторне завантаження для всіх помилок. Якщо сервер повертає 404 Not Found, то веб-сторінка не існує в даний час і той же запит навряд чи

дасть інший результат. Повний список можливих HTTP помилок визначається Internet Engineering Task Force, і доступний для перегляду на https://tools.ietf.org/html/rfc7231#section-6. У цьому документі, можна побачити, що 4xx помилки виникають, коли щось не так з нашим запитом, а 5xx помилки виникають, коли щось не так з сервером. Таким чином, ми створимо нашу функцію завантаження, яка повторятиметься у разі помилки 5xx. Ось оновлена версія для підтримки цього:

```
def download(url, num_retries=2):
      print 'Downloading:', url
      try:
            html = urllib2.urlopen(url).read()
      except urllib2.URLError as e:
            print 'Download error:', e.reason
            html = None
            if num_retries > 0:
                  if hasattr(e, 'code') and 500 <= e.code < 600:
                        # recursively retry 5xx HTTP errors
                        return download(url, num_retries-1)
      return html
```

Тепер, коли помилка завантаження зустрічається з кодом 5xx, завантаження повторяється рекурсивно викликаючи себе. Функція тепер також приймає додаткові аргумени, які вказують на то, скільки разів завантаження може бути повторене, яка встановлюється на декілька раз за замовчуванням. Ми обмежуємо число раз завантаження веб-сторінки, оскільки помилка сервера може бути нерозв'язна. Щоб перевірити цю функцію, ми можемо спробувати завантажити http://httpstat.us/500, що поверне нам код помилки – 500:

```
>>> download('http://httpstat.us/500')
Downloading: http://httpstat.us/500
Download error: Internal Server Error
Downloading: http://httpstat.us/500
Download error: Internal Server Error
Downloading: http://httpstat.us/500
Download error: Internal Server Error
```

Як і слід було очікувати, функція завантаження пробувала завантажити веб-сторінку, а потім при отриманні 500 помилки, двічі повторила спробу завантаження, перш ніж завершити роботу.

2.1.2 Налаштування агента користувача

За замовчуванням urllib2 буде завантажувати вміст з Python-urllib/2.7 агента користувача (user agent), де 2.7 це версія Python. Було б краще використовувати ідентифіковану назву агента користувача в разі, якщо проблеми виникають з нашим пошуковим роботом. Крім того, деякі сайти блокують цей агент користувача за замовчуванням, після того, як вони визначають, що програма заходить на сервер. Наприклад, на рис. 2.1 видно, що повертає сайт – http://www.meetup.com:

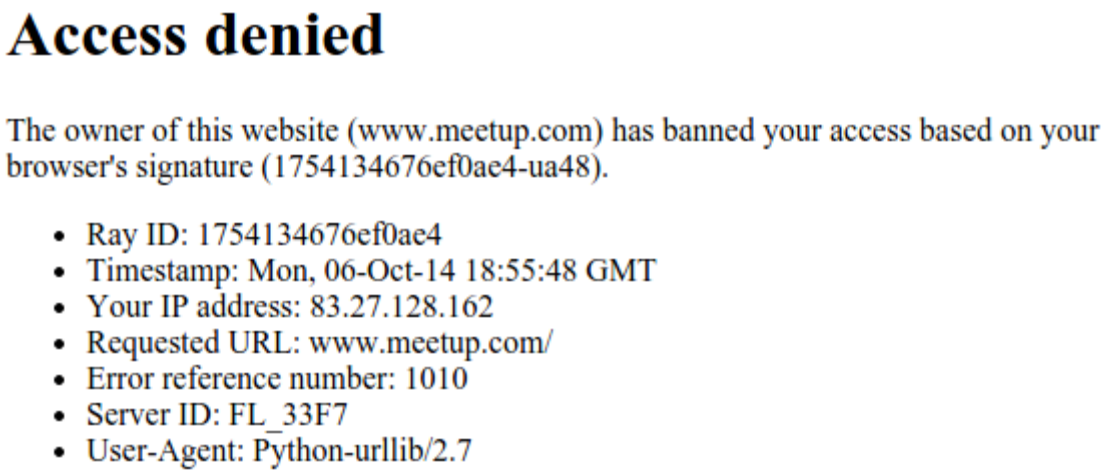


Рисунок 2.1

Таким чином, щоб надійно завантажити сторінку, нам потрібно змінити установку агента користувача. Ось оновлена версія нашої функції завантаження з агентом користувача за замовчуванням, встановленим як 'wswp' (що означає Web Scraping with Python):

```
def download(url, user_agent='wswp', num_retries=2):
      print 'Downloading:', url
      headers = {'User-agent': user_agent}
      request = urllib2.Request(url, headers=headers)
      try:
            html = urllib2.urlopen(request).read()
      except urllib2.URLError as e:
            print 'Download error:', e.reason
            html = None
            if num_retries > 0:
                  if hasattr(e, 'code') and 500 <= e.code < 600:
                        # retry 5XX HTTP errors
```

```
                        return download(url, user_agent, num_retries-1)
    return html
```

Тепер ми маємо гнучку функцію завантаження, яка може бути повторно використана в якості відновлення помилок, а саме повторного завантаження, коли це можливо, і встановлення агента користувача.

2.1.3 Підтримка проксі-сервера

Іноді необхідно отримати доступ до сайту через проксі-сервер. Наприклад, деякі веб-сайти блокують користувачів, які знаходяться за межами даної країни. Реалізувати підтримку проксі з urllib2 реально можливо таким способом:

```
proxy = ...
opener = urllib2.build_opener()
proxy_params = {urlparse.urlparse(url).scheme: proxy}
opener.add_handler(urllib2.ProxyHandler(proxy_params))
response = opener.open(request)
```

Ось оновлена версія функції завантаження, що включає це:

```
def download(url, user_agent='wswp', proxy=None, num_retries=2):
    print 'Downloading:', url
    headers = {'User-agent': user_agent}
    request = urllib2.Request(url, headers=headers)
    opener = urllib2.build_opener()
    if proxy:
        proxy_params = {urlparse.urlparse(url).scheme: proxy}
        opener.add_handler(urllib2.ProxyHandler(proxy_params))
    try:
        html = opener.open(request).read()
    except urllib2.URLError as e:
        print 'Download error:', e.reason
        html = None
        if num_retries > 0:
            if hasattr(e, 'code') and 500 <= e.code < 600:
            # retry 5XX HTTP errors
            html = download(url, user_agent, proxy, num_retries-1)
    return html
```

2.1.4 Веб-сторінки з динамічним контентом

За даними United Nations Global Audit of Web Accessibility, 73% з провідних веб-сайтів покладаються на JavaScript для важливих функціональних

можливостей. JavaScript динамічно генерує контент веб-сторінки, це може варіюватися від простих форм до SPA (Single Page Application). Наслідком цього є те, що контент багатьох веб-сторінок, який відображається в нашому веб-браузері, не доступний в оригінальному HTML, таким чином методи парсингу, не будуть працювати. В цьому пункті розглянемо варіанти виходу із цієї ситуації.

Selenium WebDriver - це програмна бібліотека для управління браузерами. Часто вживають більш коротку назву WebDriver. Іноді кажуть, що це «драйвер браузера», але насправді це ціле сімейство драйверів для різних браузерів, а також набір клієнтських бібліотек на різних мовах, що дозволяють працювати з цими драйверами. Це основний продукт, що розробляється в рамках проекту Selenium. Як вже було сказано, WebDriver - сімейство драйверів для різних браузерів плюс набір клієнтських бібліотек для цих драйверів на різних мовах програмування (рис. 2.2).

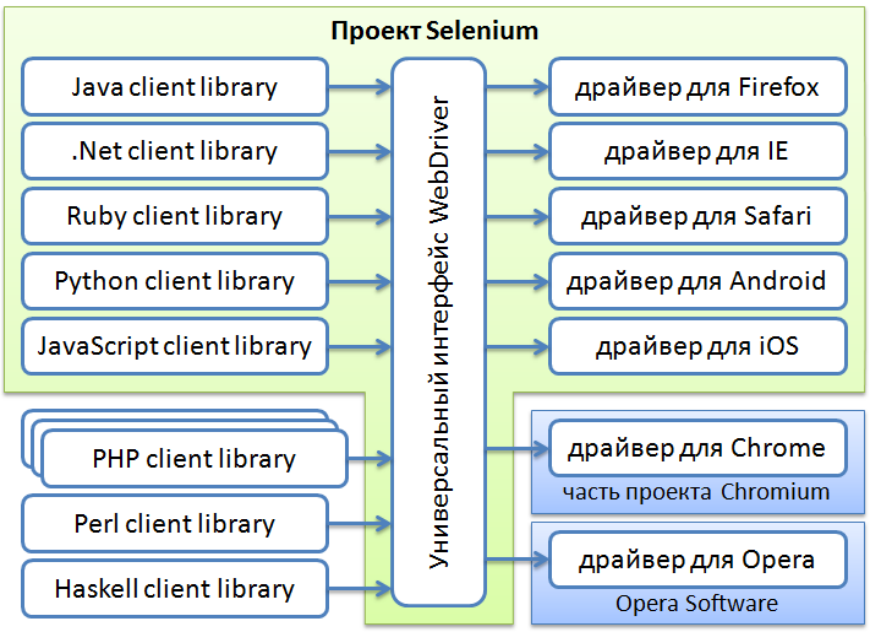


Рисунок 2.2 – Проект Selenium [4]

В рамках проекту Selenium розробляються драйвери для браузерів Firefox, Internet Explorer і Safari, а також драйвери для мобільних браузерів Android і iOS. Драйвер для браузера Google Chrome розробляється в рамках проекту Chromium, а драйвер для браузера Opera (включаючи мобільні версії)

розробляється компанією Opera Software. Тому вони формально не є частиною проекту Selenium, поширюються і підтримуються незалежно. Але звичайно, їх можна вважати частиною сімейства продуктів Selenium.

Установка Selenium:

```
pip install selenium
```

Продемонструємо як працює Selenium, перший крок полягає в створенні підключення до веб-браузеру:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.select import Select
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
driver = webdriver.Firefox()
driver.get("your url here")
# waiting for the page to load
wait = WebDriverWait(driver, 10)
wait.until(EC.visibility_of_element_located((By.ID, "content")))
data = driver.page_source
driver.close()
soup = BeautifulSoup(data, "html.parser)
```

Потім, щоб встановити, який саме елемент вибрати, можна використати ідентифікатор пошукового текстового поля. Selenium також підтримує вибір елементів за допомогою селекторів CSS або XPath. Коли визначено потрібне текстове поле, ми можемо ввести вміст у send_keys() метод, який імітує введення тексту:

```
driver.find_element_by_id('search_term').send_keys('.')
```

За допомогою Selenium можна з легкістю сфільтрувати дані чи виконати пошук по заданих критеріях, загрузити сторінку і дальше аналізувати лише корисні нам дані.

2.1.5 Взаємодія з формами

У цьому пункті розглянемо веб-сторінки, які відображають відповідний зміст, лише при введенні користувачем вхідних даних. Прикладом можуть бути будь-які форми, що відправляють дані до сервера шляхом GET і POST запитів.

Ми прикладемо мінімум зусиль за допомогою Mechanize, який забезпечує інтерфейс високого рівня для взаємодії з формами. Mechanize можна встановити за допомогою цієї команди:

```
pip install mechanize
```

Ось приклад реалізації простого завдання:

```
import mechanize
br = mechanize.Browser()
br.open(LOGIN_URL)
br.select_form(nr=0)
br['email'] = LOGIN_EMAIL
br['password'] = LOGIN_PASSWORD
response = br.submit()
br.open(COUNTRY_URL)
br.select_form(nr=0)
br['population'] = str(int(br['population']) + 1)
br.submit()
```

Цей код досить простий, нам не потрібно управляти куками, поля форм є легко доступними. Цей скрипт спочатку створює об'єкт браузера Mechanize, а потім ми переходимо до URL адреси сторінки авторизації і вибираємо поле форми. Вибрані поля форми заповнюються, передаючи ім'я і значення безпосередньо до об'єкта браузера. При налагодженні, можна викликати br.form, щоб показати стан форми, перш ніж вона буде відправлена, як показано тут:

```
print br.form
<POST http://example.webscraping.com/user/login# application/
x-www-form-urlencoded
<TextControl(email=)>
<PasswordControl(password=)>
<CheckboxControl(remember=[on])>
<SubmitControl(<None>=Login) (readonly)>
<SubmitButtonControl(<None>=) (readonly)>
<HiddenControl(_next=/) (readonly)>
<HiddenControl(_formkey=5fa268b4-0dfd-4e3f-a274-e73c6b7ce584)
(readonly)>
<HiddenControl(_formname=login) (readonly)>>
```

Взаємодія з формами є необхідним вмінням для парсингу веб-сторінок.

2.1.6 Обхід CAPTCHA

CAPTCHA, розшифровується як Completely Automated Public Turing test to tell Computers and Humans Apart. Як підказує акронім, це тест, щоб визначити, чи є користувач людиною чи ні. Типова CAPTCHA, складається із спотвореного тексту, що буде важко інтерпретувати комп'ютерній програмі, але людині розпізнати його досить не складно. Багато веб-сайтів використовують CAPTCHA, щоб спробувати запобігти від взаємодії з ботами. У цьому пункті ми розглянемо, як вирішити це непросте завдання автоматично, спочатку через Optical Character Recognition (OCR), а потім на основі API рішення.

Для подальшої роботи з CAPTCHA, перше, що нам потрібно зробити - це завантажити зображення. Для обробки зображень в Python, будем використовувати пакет Pillow. В Pillow реалізований зручний клас Image з цілою низкою методів високого рівня, що будуть використовуватися для маніпулювання CAPTCHA зображень. Ось функція, яка приймає HTML код сторінки автентифікації і повертає зображення CAPTCHA, в об'єкті Image:

```
from io import BytesIO
import lxml.html
from PIL import Image
def get_captcha(html):
      tree = lxml.html.fromstring(html)
      img_data = tree.cssselect('div#recaptcha img')[0].get('src')
      img_data = img_data.partition(',')[-1]
      binary_img_data = img_data.decode('base64')
      file_like = BytesIO(binary_img_data)
      img = Image.open(file_like)
      return img;
```

Якщо CAPTCHA не є складною, у випадку, коли текст у правильному порядку розміщений на певному фоні, то ми можемо використати OCR. OCR – це процес вилучення тексту із зображення. Ми будемо використовувати відкриту систему Tesseract OCR. Якщо оригінал CAPTCHA зображення передати в метод *pytesseract.image_to_string(img),* то ми отримаємо жахливі результати. Перш ніж правильно працювати з цією бібліотекою, ми повинні позбутися від фону зображення (рис. 2.3).

```
img.save('captcha_original.png')
```

```
gray = img.convert('L')
gray.save('captcha_gray.png')
bw = gray.point(lambda x: 0 if x < 1 else 255, '1')
bw.save('captcha_thresholded.png')
```

Рисунок 2.3 [1]

Тепер ми вже можемо визвати метод із Tesseract, який сконвертує зображення в строку.

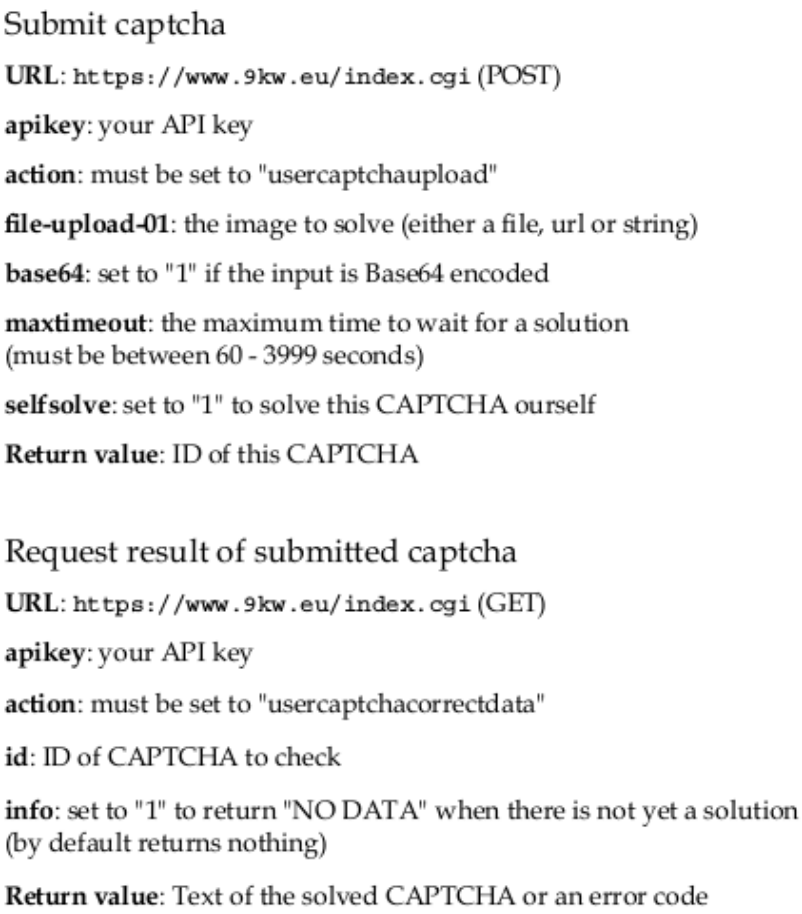
```
pytesseract.image_to_string(bw)
#'strange'
```

По даним особистого аналізу, можна сказати, протестивши 100 зображень, із них 89 були успішно розпізнані, що це досить непоганий метод обходу цієї проблеми.

У випадку, коли ми маємо складені CAPTCHA зображення (рис. 2.4), наведений вище метод не буде робочим.

Рисунок 2.4 [3]

У такому випадку можем скористатись спеціальними онлайн сервісами, що розпізнають CAPTCHA зображення, де працюють реальні люди. Спершу нам потрібно пройти реєстрацію на цьому сервісі, потім проплати послуги, наприклад 1000 зображень за 1$, далі отримуємо особистий ключ до їхньої API і вже подальша робота заключається в написанні програми, яка буде відправляти дані на сервіс та отримувати їх через визначений час. Давайте розглянемо складові API звичайного сервісу (рис. 2.5):



Submit captcha

URL: `https://www.9kw.eu/index.cgi` (POST)

apikey: your API key

action: must be set to "usercaptchaupload"

file-upload-01: the image to solve (either a file, url or string)

base64: set to "1" if the input is Base64 encoded

maxtimeout: the maximum time to wait for a solution
(must be between 60 - 3999 seconds)

selfsolve: set to "1" to solve this CAPTCHA ourself

Return value: ID of this CAPTCHA

Request result of submitted captcha

URL: `https://www.9kw.eu/index.cgi` (GET)

apikey: your API key

action: must be set to "usercaptchacorrectdata"

id: ID of CAPTCHA to check

info: set to "1" to return "NO DATA" when there is not yet a solution
(by default returns nothing)

Return value: Text of the solved CAPTCHA or an error code

Рисунок 2.5

Наступна задача – написати програму, яка буде відправляти зображення на сервіс і отримувати результат за допомогою POST і GET запитів відповідно.

2.2 Аналіз та обробка даних

Отже, вище ми розглянули всі можливі варіанти для завантаження веб-сторінок. Тепер нам потрібно шляхом вилучення отримати лише потрібні нам

дані. У даному підрозділі розглянемо структуру веб-сторінок. Розглянемо три підходи для отримання даних: регулярні вирази, Beautiful Soup і lxml. І вкінці порівняємо ці методи по різним критеріям.

2.2.1 Аналіз структури веб-сторінок

Щоб зрозуміти як влаштована веб-сторінка, розглянемо її вихідний код за допомогою браузера (рис.2.6).

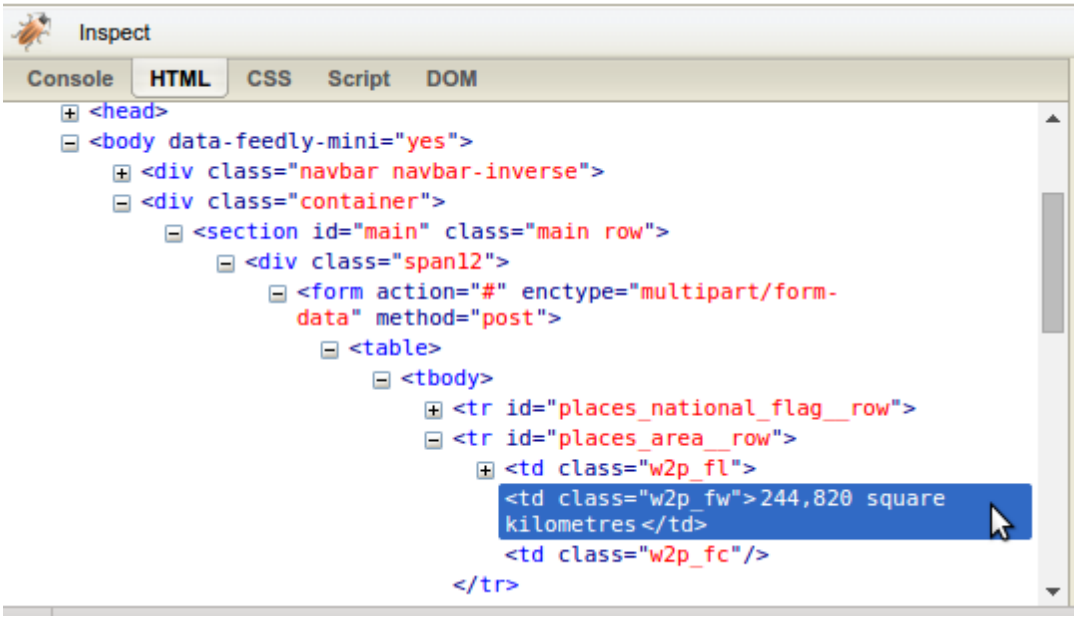


Рисунок 2.6 – Ієрархія HTML-документа

Оцінивши картину, стає зрозуміло, що структура HTML досить проста – складається з HTML-тегів та, безпосередньо, контенту.

2.2.2 Три підходи обробки веб-сторінок

Тепер, коли ми вже ознайомились із структурою веб-сторінок, розглянемо по порядку три абсолютно різні підходи парсингу даних:

1. регулярні вирази;
2. BeautifulSoup модуль;
3. lxml модуль.

Більш детально ознайомитись з можливостями **регулярних виразів** можна на https://docs.python.org/2/howto/regex.html. Для того, щоб спарсити область, використовуючи регулярні вирази, продемонструємо приклад пошуку вмісту елемента <tr>, в такий спосіб:

```
re.findall('<tr
id="places_area__row">.*?<td\s*class=["\']w2p_fw["\']>(.*?)
</td>', html)
['244,820 square kilometres']
```

Регулярні вирази важко побудувати, так як вони нечитабельні. Крім того, є ще й інші незначні нюанси, які порушили б роботу програми, наприклад, якщо до тегу <tr> розробники додадуть атрибут title, і на цьому наша функція вже не буде викновутись, потрібно заново переписувати регулярний вираз. З цього прикладу видно, що регулярні вирази забезпечують швидкий спосіб відокремлення даних, але дуже крихкі і легко ламаються, коли веб-сторінка оновлюється. На щастя, є більш ефективні рішення.

BeautifulSoup є популярним модулем, який аналізує веб-сторінки, а також надає зручний інтерфейс для навігації контенту. Остання версія може бути встановлена за допомогою наступної команди:

```
pip install beautifulsoup4
```

Перший крок з BeautifulSoup – це перетворення завантаженого HTML-документа в soup-документ. Більшість веб-сторінок не містять абсолютно правильного HTML-коду і BeautifulSoup прекрасно з цим справляється за допомогою встроєного функціоналу. Наприклад, розробник забув закрити атрибути відповідними тегами, для цього в BeautifulSoup існує метод *soup.prettify(),* який дозволяє згенерувати абсолютно правильний код. BeautifulSoup в змозі правильно інтерпретувати відсутні лапки атрибутів і закриваючі теги, а також додати <html> і <body> теги, щоб сформувати повний HTML-документ. Тепер ми можемо перейти до елементів, які ми хочемо використовувати за допомогою find () і find_all () методів:

```
from bs4 import BeautifulSoup
url = 'http://example.webscraping.com/places/view/United-Kingdom-239'
html = download(url)
soup = BeautifulSoup(html)
```

```
# locate the area row
tr = soup.find(attrs={'id':'places_area__row'})
td = tr.find(attrs={'class':'w2p_fw'}) # locate the area tag
area = td.text # extract the text from this tag
print area
[244,820 square kilometres]
```

За об’ємом цей код більший, ніж регулярні вирази, але простіший в реалізації і зрозумілий. Крім того, нам більше не потрібно турбуватися про проблеми в незначних оновленнях структури, так як додаткові пробіли або атрибути тега, не впливатимуть на результат роботи програми.

LXML є Python обгорткою поверх libxml2 XML-бібліотеки, написаної на C, яка допомагає зробити це швидше, ніж Beautiful Soup, але складніше в плані установки на деякі ПК. Останні інструкції по установці доступні на http://lxml.de/installation.html. Як і з Beautiful Soup, першим кроком є перетворення потенційно недійсних HTML в правильний формат:

```
fixed_html = lxml.html.tostring(tree, pretty_print=True)
```

Ось приклад використання lxml CSS селекторів для роботи з даними:

```
tree = lxml.html.fromstring(html)
td = tree.cssselect('tr#places_area__row > td.w2p_fw')[0]
area = td.text_content()
print area
[244,820 square kilometres]
```

CSS селектори - це шаблони, які використовуються для вибору елементів. Ось деякі приклади загальних селекторів, які вам будуть потрібними (рис. 2.7):

```
Select any tag: *
Select by tag <a>: a
Select by class of "link": .link
Select by tag <a> with class "link": a.link
Select by tag <a> with ID "home": a#home
Select by child <span> of tag <a>: a > span
Select by descendant <span> of tag <a>: a span
Select by tag <a> with attribute title of "Home": a[title=Home]
```

Рисунок 2.7 [11]

Запустивши тестові приклади, знаходимо час, який знадобиться для виконання однієї і тієї самої задачі, реалізованої різними шляхами.

```
regex_test took 40.032 ms
bs_test took 54206.303 ms
lxml_test took 1863.463 ms
```

Проаналізувавши та дослідивши три підходи обробки веб-сторінок, можемо побудувати таблицю 2.1, яка включає в себе переваги та недоліки кожного методу.

Таблиця 2.1 – Оцінка підходів обробки даних

Підхід	Швидкодія	Простота у використанні	Простота в установці
Regular expressions	Висока	Складно	Легко
BeautifulSoup	Низька	Легко	Легко
Lxml	Висока	Легко	Складно

Отже, якщо потрібно спарсити невелекі об’єми даних, можна сміло використовувавти BeautifulSoup модуль. Або використовувати регулярні вирази, якщо потрібно вказати більш розширену унікальність критеріїв відбору даних. В загальному lxml – це найкращий вибір для розробки автоматизованих систем парсингу, тому що він швидкий і надійний, а також не є складним у використанні.

2.3 Експорт даних у БД

Після обробки даних, наступне завдання – правильно зберегти отримані дані і відобразити їх користувачеві на сайті http://dl-cloud.kpi.ua. Отже, нам потрібно зберігати велику кількість даних, а так як складних операцій з БД не передбачається, будемо використовувати NoSQL бази даних, які простіші в масштабуванні від традиційних реляційних БД. Зокрема, вибір впав на MongoDB, яка в даний час є найпопулярнішою базою даних NoSQL.

NoSQL розшифровується як Not Only SQL і являє собою відносно новий підхід до дизайну бази даних. Традиційна реляційна модель, використовує фіксовану схему і розбиває дані в таблиці. Проте, з великими наборами даних, коли дані занадто великі для одного сервера, виникає потреба в масштабуванні на декількох серверах. Це не дуже добре узгоджується з реляційною моделлю

тому що, при запиті декількох таблиць, дані не завжди будуть доступні на цьому ж сервері. Системи управління базами даних NoSQL надають механізм збереження та отримання даних, що організовані в інакший спосіб, ніж звичний реляційний підхід, хоча окремі NoSQL системи можуть надавати SQL-подібний синтаксис. Технологія NoSQL розроблена для забезпечення простоти архітектури бази даних та горизонтального масштабування. Ці якості досягаються за рахунок принципів організації структур даних (ключ-значення, граф, чи документ), які істотно відрізняються від СКРБД (RDBMS). Як наслідок, деякі операції доступу до даних виконуються швидше в NoSQL (наприклад такі, що передбачають складні операції реляційної алгебри), а деякі — в RDBMS. NoSQL бази даних використовують в індустрії високонавантажених веб-додатків реального часу, хоча багато NoSQL сховищ не гарантують цілісність даних (CAP теорема), хоча забезпечують їх високу доступність та реплікації.

MongoDB – доволі швидка об'єктно-документарна база даних. Ідеологічно це якийсь симбіоз між звичною реляційною БД і “ключ-значення” сховищем, і на мій погляд, вельми вдалий. З одного боку, вона дозволяє робити дуже швидкі операції над об'єктом, знаючи його ідентифікатор, а з іншого, надає потужний інструмент для складних взаємодій.

- Колекція - іменована множина об'єктів, при цьому один об'єкт належить лише одній колекції.
- Об'єкт - сукупність властивостей, включаючи унікальний ідентифікатор _id.
- Властивість - сукупність назви і відповідного йому типу і значення.
- Типи властивостей - рядок, ціле число, число з плаваючою точкою, масив, об'єкт, бінарний рядок, байт, символ, дата, boolean, null.
- Підтримуються операції вибірки (count, group, MapReduce ...), вставки, зміни та видалення. Зв'язків між об'єктами немає, об'єкти можуть лише зберігати інші об'єкти у властивостях. Підтримуються як унікальні, так і

композитні індекси. Індекси можна накладати на властивості вкладених об'єктів.

- Підтримується реплікація, реалізований fail-over.
- Реалізовано MapReduce і шардінг.
- Оскільки об'єкти можуть мати довільний набір властивостей, для каталогу досить створити колекцію і складати туди об'єкти. При цьому пошук буде вестися за індексами.

Переваги MongoDB яскраво виражені на insert, вони відбуваються ну дуже швидко. До речі, формат зберігання і формат передачі об'єктів по мережі один і той же, так що для вибірки якогось об'єкта треба всього лише знайти його позицію за індексом і повернути клієнту частину файлу певної довжини - ніякої абстракції над підсистемою збереження даних. В якості унікального ідентифікатора використовується не поле auto-increment, а 12-байтне унікальне число, що генерується на клієнті. Таким чином, по-перше немає проблеми з синхронізацією реплік, тобто можна незалежно робити вставки на дві різні машини, і конфлікту не виникне. По-друге, не буде конфліктів з переповненням цілого числа, ну і після перевтілення бази даних, пошукові системи не будуть адресувати на нові статті за старими посиланнями.

2.4 Висновки

У даному розділі було розглянуто можливості Python для створення веб-парсера. А саме, розглянуто проблеми загрузки веб-сторінок, варіанти їх вирішення. Досліджені готові методи та їх алгоритми для аналізу та обробки html документа, їх порівняльна характеристика. Обрано найбільш оптимальне рішення для нашої програми. Розглянуто принципи NoSQL БД як засобу для збереження вихідних даних.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1 Проектування розробки програми

3.1.1 Діаграма станів

Діаграма станів - орієнтований граф для кінцевого автомата, в якому:

- вершини позначають стан
- дуги показують переходи між двома станами.

Спроектуємо діаграму станів програми-парсера (рис. 3.1 – Діаграма станів).

Рисунок 3.1 – Діаграма станів

3.1.2 IDEF0 діаграма

IDEF0 використовується для створення функціональної моделі (рис. 3.2), що відображає структуру і функції системи, а також потоки інформації і матеріальних об'єктів, що зв'язують ці функції.

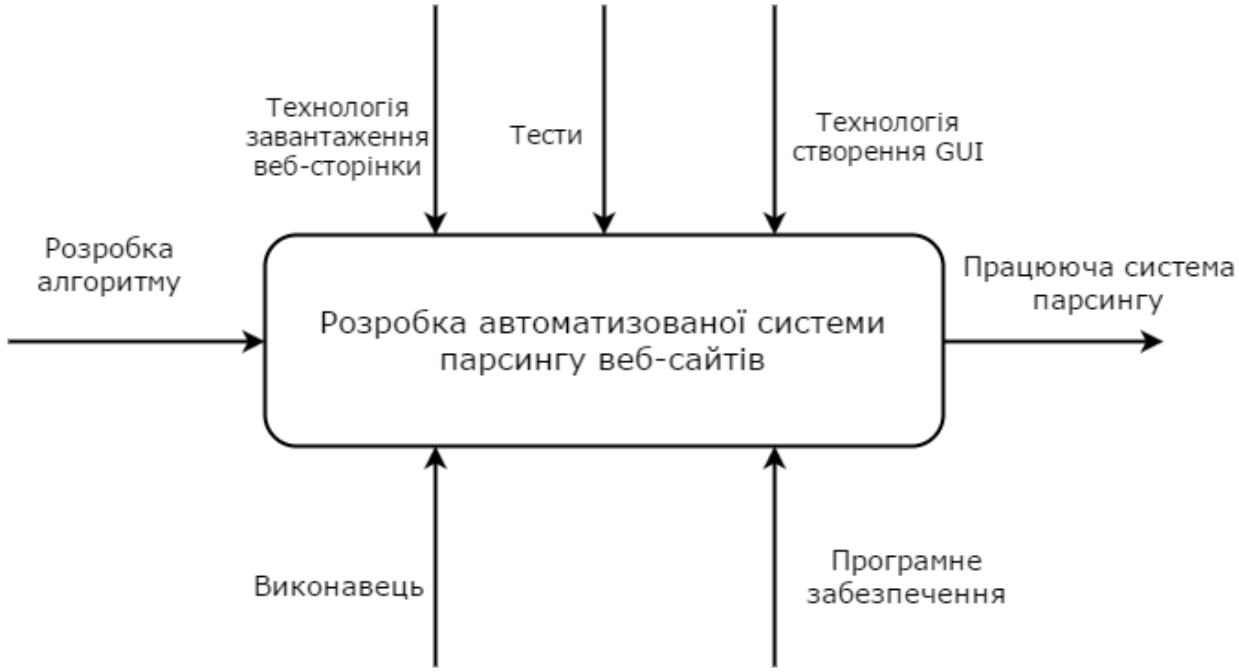


Рисунок 3.2 – IDEF0 діаграма

3.1.3 Діаграма потоків даних

Діаграма потоків даних (англ. *Data Flow Diagram*) — модель проектування, графічне представлення «потоків» даних в інформаційній системі. Діаграма потоків даних також може використовуватись для візуалізації процесів обробки даних.

Діаграми потоків даних містять чотири типи графічних елементів:

- процеси - являють собою трансформацію даних в рамках описуваної системи;
- сховища даних (репозиторії);
- зовнішні по відношенню до системи сутності;
- потоки даних між елементами трьох попередніх типів.

Спроектуємо DFD діаграму для нашого завдання (рис. 3.3).

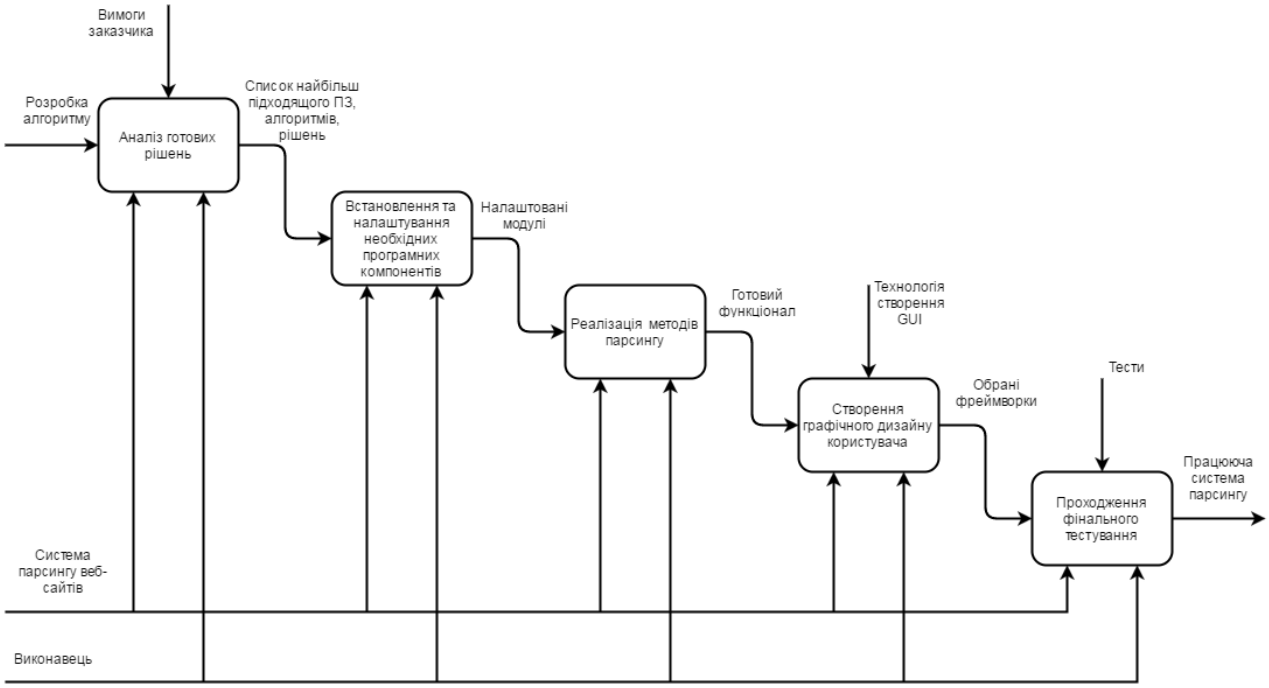


Рисунок 3.3 – DFD діаграма

3.1.4 Діаграма прецедентів

Діаграма прецедентів візуально зображає різноманітні сценарії взаємодії між акторами (користувачами) і прецедентами (випадками використання), описує функціональні аспекти системи (бізнес логіку).

Діаграми прецедентів відіграють важливу роль не тільки у комунікації між збирачами вимог до проекту і потенційними користувачами. Діаграми прецедентів дописані бізнес логікою і детальними специфікаціями прецедентів, як джерельна інформація, успішно використовують учасники розробки проекту на всіх його фазах (зародження, дизайн, програмування, тестування, документування). Добре продумані і завершені специфікації прецедентів легко перевтілюються у тестові випадки.

Елементи діаграми прецедентів:

- Актор – користувач.
- Прецедент – випадок використання, дія. Позначається овалом.
- Граничні межі системи охоплюють усі випадки використання у системі. Позначається прямокутником.

Елементи взаємодії діаграми прецедентів:

- Використовує – користувач виконує дію.
- Включає – один прецедент використовує іншого.
- Розширює – представлення дочірніх прецедентів.
- Вимагає – наступний прецедент вимагає виконання попереднього.
- Схожий – прецеденти подібні, але описують різну функціональність.
- Рівнозначний - подібна функціональність, але користувач сприймає, як різну.

Спроектуємо діаграму перецедентів (рис. 3.4).

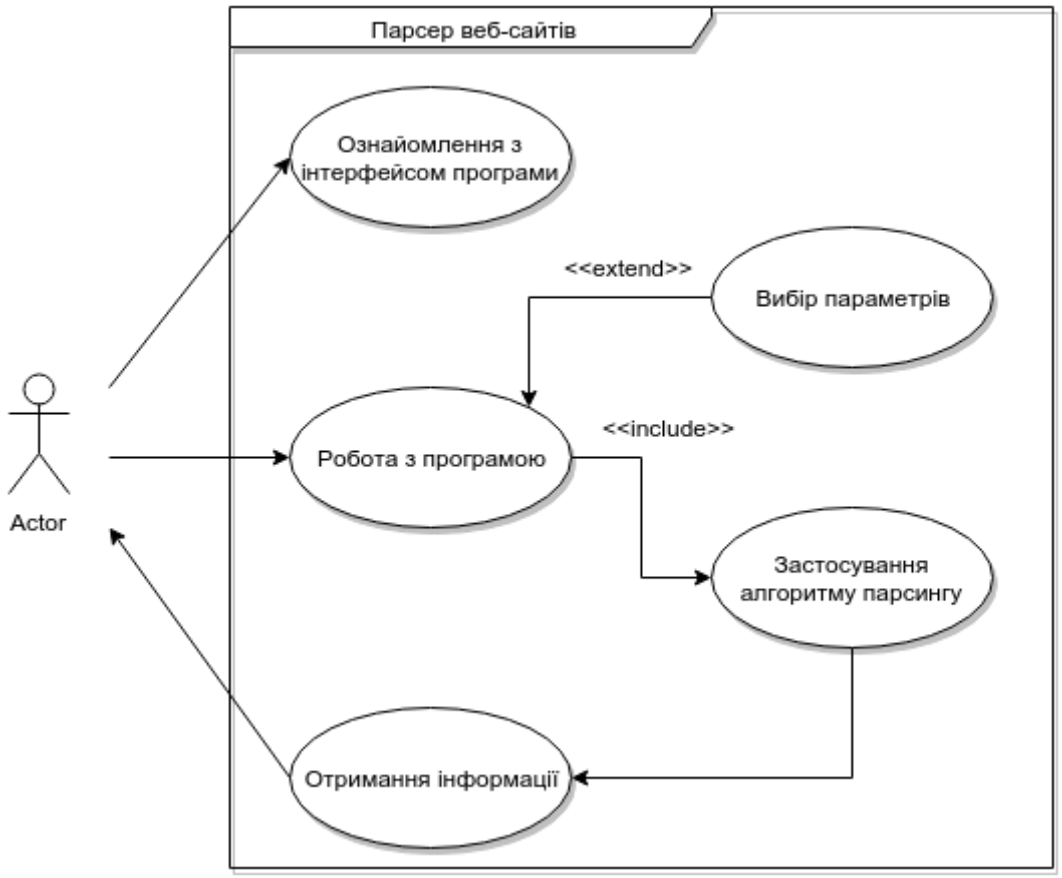


Рисунок 3.4 – Діаграма прецедентів

3.2 Розробка функціоналу програми

3.2.1 Реалізація методу завантаження веб-сторінки

Для того, щоб уникнути необхідності передавати різні параметри для кожного завантаження, ми скористаємось можливістю реорганізувати функцію завантаження, описану в другому розділі в клас, так щоб параметри встановлювались один раз в конструкторі і повторно використовувались кілька разів. Ось оновлена реалізація для підтримки цього:

```
import urlparse
import urllib2
import random
import time
from datetime import datetime, timedelta
import socket


DEFAULT_AGENT = 'wswp'
DEFAULT_DELAY = 5
DEFAULT_RETRIES = 1
DEFAULT_TIMEOUT = 60


class Downloader:
    def __init__(self, delay=DEFAULT_DELAY, user_agent=DEFAULT_AGENT,
      proxies=None, num_retries=DEFAULT_RETRIES, timeout=DEFAULT_TIMEOUT,
      opener=None, cache=None):
        socket.setdefaulttimeout(timeout)
        self.throttle = Throttle(delay)
        self.user_agent = user_agent
        self.proxies = proxies
        self.num_retries = num_retries
        self.opener = opener
        self.cache = cache


    def __call__(self, url):
        result = None
        if self.cache:
            try:
                result = self.cache[url]
            except KeyError:
                # url is not available in cache
                pass
            else:
                if self.num_retries > 0 and 500 <= result['code'] < 600:
                    # server error so ignore result from cache and re-download
                    result = None
        if result is None:
```

```
            # result was not loaded from cache so still need to download
            self.throttle.wait(url)
            proxy = random.choice(self.proxies) if self.proxies else None
            headers = {'User-agent': self.user_agent}
            result = self.download(url, headers, proxy=proxy,
            num_retries=self.num_retries)
            if self.cache:
                # save result to cache
                self.cache[url] = result
        return result['html']


    def download(self, url, headers, proxy, num_retries, data=None):
        print 'Downloading:', url
        request = urllib2.Request(url, data, headers or {})
        opener = self.opener or urllib2.build_opener()
        if proxy:
            proxy_params = {urlparse.urlparse(url).scheme: proxy}
            opener.add_handler(urllib2.ProxyHandler(proxy_params))
        try:
            response = opener.open(request)
            html = response.read()
            code = response.code
        except Exception as e:
            print 'Download error:', str(e)
            html = ''
            if hasattr(e, 'code'):
                code = e.code
                if num_retries > 0 and 500 <= code < 600:
                    # retry 35XX HTTP errors
                    return self._get(url, headers, proxy, num_retries-1, data)
            else:
                code = None
        return {'html': html, 'code': code}

class Throttle:
    """Throttle downloading by sleeping between requests to same domain
    """
    def __init__(self, delay):
        # amount of delay between downloads for each domain
        self.delay = delay
        # timestamp of when a domain was last accessed
        self.domains = {}

    def wait(self, url):
        """Delay if have accessed this domain recently
        """
        domain = urlparse.urlsplit(url).netloc
        last_accessed = self.domains.get(domain)
        if self.delay > 0 and last_accessed is not None:
            sleep_secs = self.delay - (datetime.now() - last_accessed).seconds
            if sleep_secs > 0:
                time.sleep(sleep_secs)
        self.domains[domain] = datetime.now()
```

3.2.2 Реалізація методу парсингу веб-документа

Реалізуємо метод парсингу веб-сторінки за допомогою функцій бібліотеки lxml. Використовуємо даний модуль, адже саме він найшвидше справляється у нашому випадку з пошуком інформації по веб-документі. Так як в ньому встроєнні методи, що дозволяють розкласти html-сторінку в деревоподібну структуру і здійснювати пошук по ній за допомогою рекурсії. Розглянемо приклад функції для парсингу одного з сайтів:

```
def parseUdacity():
      results = dict(zip(["android", "data-science", "georgia-tech-masters-in-
            cs", "ios", "non-tech", "software-engineering", "web-development"],
            [[], [], [], [], [], [], []]))
      startUrl = "https://www.udacity.com"
      for key in results:
            tree = initParser(startUrl, "/courses/" + str(key))
            name = (tree.xpath('//a[@data-course-title=""]/text()'))
            href = tree.xpath('//a[@data-course-title=""]/@href')
            for index in range(len(href)):
                  results[key].append(name[index].replace('\n                ',
                  '').replace('\n              ', ''))
                  results[key].append(createUrl(startUrl, href[index]))
      return results;
```

Для наглядності збережемо результати у CSV файл (рис. 3.5):

	A	B	C
1	lessons_name	lessons_url	category
2	How to Use Git and GitHub	https://www.udacity.com/course/how-to-use-git-and-github--ud775	software-engineering
3	Intro to Computer Science	https://www.udacity.com/course/intro-to-computer-science--cs101	software-engineering
4	Intro to Java Programming	https://www.udacity.com/course/intro-to-java-programming--cs046	software-engineering
5	Programming Foundations with Python	https://www.udacity.com/course/programming-foundations-with-python--ud036	software-engineering
6	Intro to Hadoop and MapReduce	https://www.udacity.com/course/intro-to-hadoop-and-mapreduce--ud617	software-engineering
7	Software Development Process	https://www.udacity.com/course/software-development-process--ud805	software-engineering
8	Artificial Intelligence for Robotics	https://www.udacity.com/course/artificial-intelligence-for-robotics--cs373	software-engineering
9	Data Wrangling with MongoDB	https://www.udacity.com/course/data-wrangling-with-mongodb--ud032	software-engineering
10	Intro to Parallel Programming	https://www.udacity.com/course/intro-to-parallel-programming--cs344	software-engineering
11	Intro to Artificial Intelligence	https://www.udacity.com/course/intro-to-artificial-intelligence--cs271	software-engineering
12	Interactive 3D Graphics	https://www.udacity.com/course/interactive-3d-graphics--cs291	software-engineering
13	Software Testing	https://www.udacity.com/course/software-testing--cs258	software-engineering
14	Design of Computer Programs	https://www.udacity.com/course/design-of-computer-programs--cs212	software-engineering
15	Intro to Algorithms	https://www.udacity.com/course/intro-to-algorithms--cs215	software-engineering
16	Programming Languages	https://www.udacity.com/course/programming-languages--cs262	software-engineering
17	Applied Cryptography	https://www.udacity.com/course/applied-cryptography--cs387	software-engineering
18	Software Debugging	https://www.udacity.com/course/software-debugging--cs259	software-engineering
19	Intro to Theoretical Computer Science	https://www.udacity.com/course/intro-to-theoretical-computer-science--cs313	software-engineering
20	Machine Learning: Unsupervised Learning	https://www.udacity.com/course/machine-learning-unsupervised-learning--ud741	software-engineering
21	Networking for Web Developers	https://www.udacity.com/course/networking-for-web-developers--ud256	software-engineering
22	Scalable Microservices with Kubernetes	https://www.udacity.com/course/scalable-microservices-with-kubernetes--ud615	software-engineering
23	Technical Interview	https://www.udacity.com/course/technical-interview--ud513	software-engineering
24	How to Build a Startup	https://www.udacity.com/course/how-to-build-a-startup--ep245	non-tech
25	Product Design	https://www.udacity.com/course/product-design--ud509	non-tech
26	App Monetization	https://www.udacity.com/course/app-monetization--ud518	non-tech
27	Rapid Prototyping	https://www.udacity.com/course/rapid-prototyping--ud723	non-tech
28	App Marketing	https://www.udacity.com/course/app-marketing--ud719	non-tech
29	Statistics	https://www.udacity.com/course/statistics--st095	non-tech
30	Intro to the Design of Everyday Things	https://www.udacity.com/course/intro-to-the-design-of-everyday-things--design101	non-tech

data

Рисунок 3.5 – Результат роботи програми

Приклад коду програми для запису даних в CSV файл.

```
with open('data.csv', 'w') as csvfile:
    fieldnames = ['lessons_name', 'lessons_url', 'category']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
    writer.writeheader()
    for key in data_array:
      index = 0
      while index < len(data_array[key]):
            urlIndex = index + 1
            writer.writerow({'lessons_name': data_array[key][index],
             'lessons_url': data_array[key][index + 1], 'category': key})
            index = index + 2
```

3.3 Розробка графічного інтерфейсу програми

Графічний інтерфейс користувача (англ. GUI, Graphical user interface) – повинен бути простим і легко доступним для кожного користувача. Було прийнято рішення, що програма повинна бути десктопною і кросплатформеною. Даний інтерфейс програми (рис. 3.6) розроблений на спеціальному фреймворку для Python – Kivy.

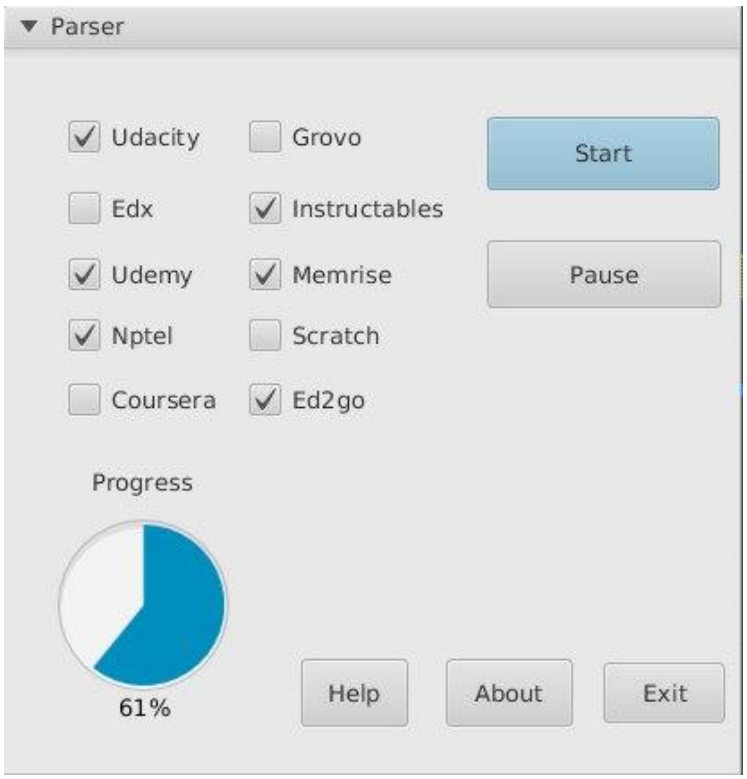


Рисунок 3.6

Kivy – кросплатформене вільне програмне забезпечення. Графічний движок побудований на OpenGL ES 2, використовуючи сучасний і швидкий графічний конвеєр. Інструментарій поставляється з більш ніж 20 віджетів, все легко розширюється. Багато частин написані на мові C за допомогою Cython, і протестовані з регресійних тестів.

3.4 Висновки

У даному розділі розглянуто структуру програми, спроектовані такі діаграми: діаграма станів, діаграма потоків даних, діаграма перецедентів, IDEF0 діагарама. Також частково розглянуто функціонал програми, основні методи. Ознайомлено з інтерфейсом парсера. Перевірено достовірність результатів роботи програми.

4. ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проводиться оцінка основних характеристик програмного продукту, Cloud Parser. Інтерфейс користувача був розроблений за допомогою мов програмування HTML, CSS, JavaScript, Python у середовищі розробки WebStorm.

Програмний продукт призначено для адміністратора сайта, з метою полегшення заповнення контента сайта під управлінням будь-якої операційної системи.

Нижче наведено аналіз різних варіантів реалізації модулю з метою вибору оптимальної, з огляду при цьому як на економічні фактори, так і на характеристики продукту, що впливають на продуктивність роботи і на його сумісність з апаратним забезпеченням. Для цього було використано апарат функціонально-вартісного аналізу.

Функціонально-вартісний аналіз (ФВА) – це технологія, яка дозволяє оцінити реальну вартість продукту або послуги незалежно від організаційної структури компанії. Як прямі, так і побічні витрати розподіляються по продуктам та послугам у залежності від потрібних на кожному етапі виробництва обсягів ресурсів. Виконані на цих етапах дії у контексті метода ФВА називаються функціями.

Мета ФВА полягає у забезпеченні правильного розподілу ресурсів, виділених на виробництво продукції або надання послуг, на прямі та непрямі витрати. У даному випадку – аналізу функцій програмного продукту й виявлення усіх витрат на реалізацію цих функцій.

Фактично цей метод працює за таким алгоритмом:

– визначається послідовність функцій, необхідних для виробництва продукту. Спочатку – всі можливі, потім вони розподіляються по двом групам: ті, що впливають на вартість продукту і ті, що не впливають. На цьому ж етапі

оптимізується сама послідовність скороченням кроків, що не впливають на цінність і відповідно витрат.

– для кожної функції визначаються повні річні витрати й кількість робочих часів.

– для кожної функції наоснові оцінок попереднього пункту визначається кількісна характеристика джерел витрат.

– після того, як для кожної функції будуть визначені їх джерела витрат, проводиться кінцевий розрахунок витрат на виробництво продукту.

4.1 Постановка задачі техніко-економічного аналізу

У роботі застосовується метод ФВА для проведення техніко-економічний аналізу розробки.

Відповідно цьому варто обирати і систему показників якості програмного продукту.

Технічні вимоги до продукту наступні:

– програмний продукт повинен функціонувати на персональних комп’ютерах із стандартним набором компонент;

– забезпечувати високу швидкість обробки великих об’ємів даних у реальному часі;

– забезпечувати зручність і простоту взаємодії з користувачем або з розробником програмного забезпечення у випадку використовування його як модуля;

– передбачати мінімальні витрати на впровадження програмного продукту.

4.1.1 Обгрунтування функцій програмного продукту

Головна функція F_0– розробка програмного продукту, який аналізує процес за вхідними даними та будує його модель для подальшого

прогнозування. Виходячи з конкретної мети, можна виділити наступні основні функції ПП:

F_1 – вибір мови програмування;

F_2 – вибір оптимальної СКБД;

F_3 – інтерфейс користувача.

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F_1:

а) мова програмування Python;

б) мова програмування JavaScript;

Функція F_2:

а) MongoDB;

б) PostgreSQL.

Функція F_3:

а) інтерфейс користувача, створений за допомогою Kivy;

б) інтерфейс користувача, створений за допомогою HTML, CSS.

4.1.2 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведені у морфологічній карті системи (рис. 4.1). На основі цієї карти побудовано позитивно-негативну матрицю варіантів основних функцій (таблиця 4.1).

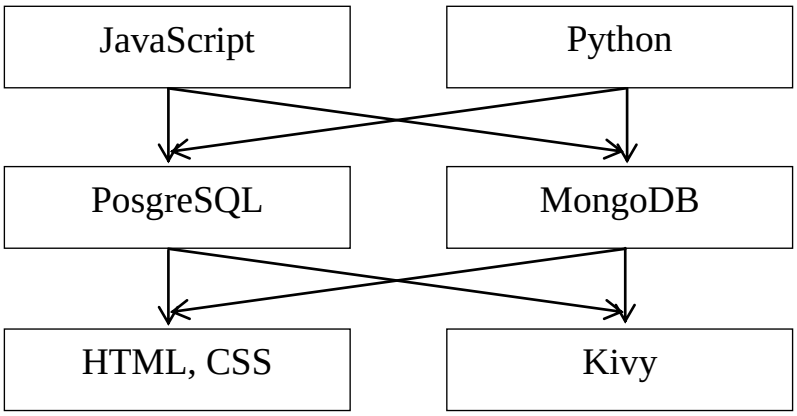


Рисунок 4.1 – Морфологічна карта

Морфологічна карта відображує всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів ПП.

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому, що вони не відповідають поставленим перед програмним продуктом задачам. Ці варіанти відзначені у морфологічній карті. Функція *F1:*

Оскільки розрахунки проводяться з великими об'ємами вхідних даних, то час виконання програмного коду є дуже необхідним, тому варіант а) має бути відкинутий.

Функція *F2:*

Вибір СКБД не відіграє велику роль у даному програмному продукту, тому вважаємо варіанти а) та б) гідними розгляду.

Таблиця 4.1 – Позитивно-негативна матриця

Основні функції	**Варіанти реалізації**	**Переваги**	**Недоліки**
F1	*А*	Існує багато готових рішень	Висока швидкодія
	Б	Займає менше часу при написанні коду	Більший час на виконання операцій
F2	А	Швидко виконуються операції	Відсутність вкладених запитів
	Б	Надійність, внесення змін без перезапуску, безкоштовність	Необхідність додаткової інсталяції, низький рівень користувацької підтримки
F3	*А*	Простота створення	Витрачається багато часу на написання коду
	Б	Простота створення	Витрачається багато часу на вивчення фреймворку

Функція *F3:*

Оскільки, програмний продукт реалізується на мові Python, використовуємо варіант А як єдиний можливий.

Таким чином, будемо розглядати такі варіанти реалізації ПП:

1. F1а – F2а – F3а

2. F1б – F2б – F3б

Для оцінювання якості розглянутих функцій обрана система параметрів, описана нижче.

4.2 Обгрунтування системи параметрів пп

4.2.1 Опис параметрів

На підставі даних про основні функції, що повинен реалізувати програмний продукт, вимог до нього, визначаються основні параметри виробу, що будуть використані для розрахунку коефіцієнта технічного рівня.

Для того, щоб охарактеризувати програмний продукт, будемо використовувати наступні параметри:

– *X1* – швидкодія мови програмування;

– *X2* – об’єм пам’яті для збереження даних;

– *X3* – час обробки даних;

– *X4* – потенційний об’єм програмного коду.

X1: Відображає швидкодію операцій залежно від обраної мови програмування.

X2: Відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних під час виконання програми.

X3: Відображає час, який витрачається на дії.

X4: Показує розмір програмного коду який необхідно створити безпосередньо розробнику.

4.2.2 Кількісна оцінка параметрів

Гірші, середні і кращі значення параметрів вибираються на основі вимог замовника й умов, що характеризують експлуатацію ПП як показано у табл. 4.2.

Таблиця 4.2 – Основні параметри ПП

Назва Параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			Гірші	середні	кращі
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки запитів користувача	X3	мс	1000	420	60
Потенційний об’єм програмного коду	X4	кількість строк коду	2000	1500	1000

За даними таблиці 4.2 будуються графічні характеристики параметрів – рис. 4.2 – рис. 4.5.

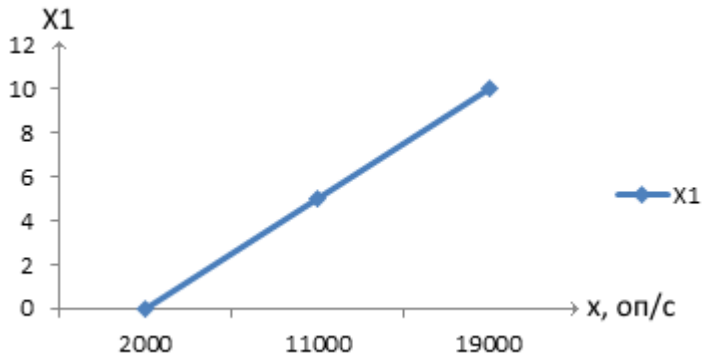


Рисунок 4.2 – X1, швидкодія мови програмування

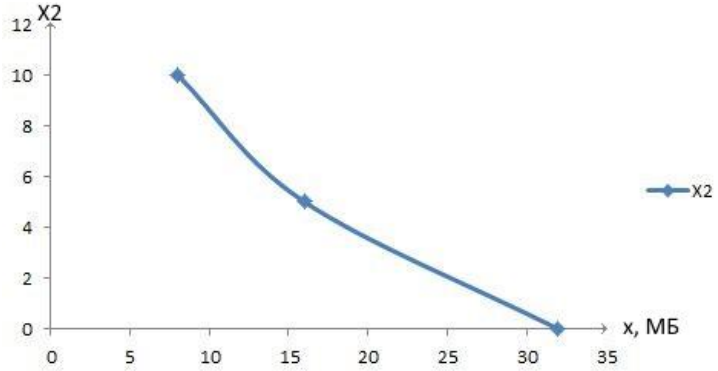


Рисунок 4.3 – X2, об’єм пам’яті для збереження даних

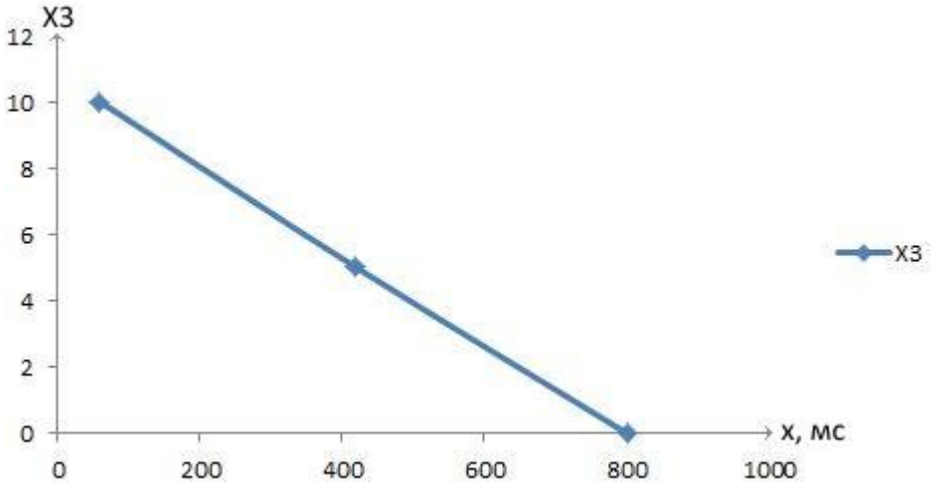


Рисунок 4.4 – X3, час виконання запитів користувача

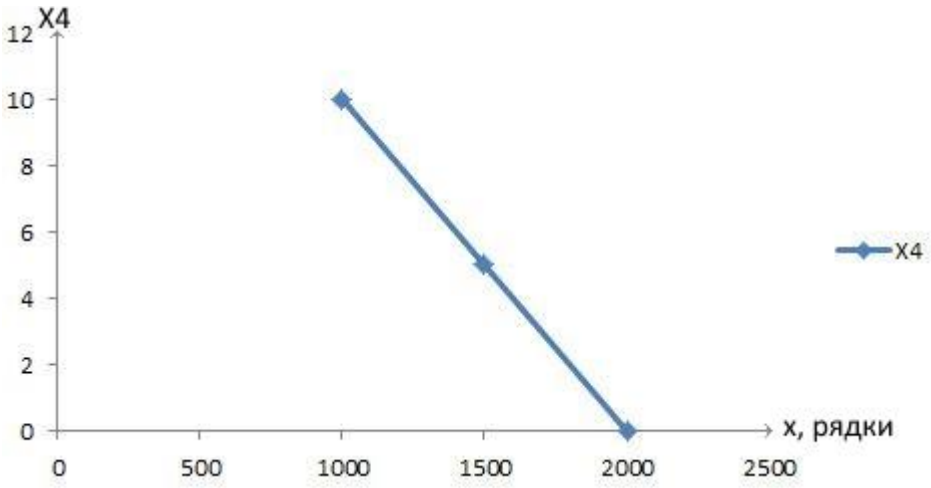


Рисунок 4.5 – X4, потенційний об’єм програмного коду

4.2.3 Аналіз експертного оцінювання параметрів

Після детального обговорення й аналізу кожний експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні параметрів моделей адаптивного прогнозування і обчислення прогнозних значень.

Значимість кожного параметра визначається методом попарного порівняння. Оцінку проводить експертна комісія із 7 людей. Визначення

коефіцієнтів значимості передбачає:

– визначення рівня значимості параметра шляхом присвоєння різних рангів;

– перевірку придатності експертних оцінок для подальшого використання;

– визначення оцінки попарного пріоритету параметрів;

– обробку результатів та визначення коефіцієнту значимості.

Результати експертного ранжування наведені у таблиці 4.3.

Таблиця 4.3 – Результати ранжування параметрів

Позначення параметра	Назва параметра	Одиниці виміру	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	Δ_i^2
			1	2	3	4	5	6	7			
X1	Швидкодія мови програмування	Оп/мс	4	3	4	4	4	4	4	27	0,75	0,56
X2	Об’єм пам’яті для збереження даних	Мб	4	4	4	3	4	3	3	25	-1,25	1,56
X3	Час обробки запитів користувача	Мс	2	2	1	2	1	2	2	12	-14,25	203,06
X4	Потенційний об’єм програмного коду	кількість строк коду	5	6	6	6	6	6	6	41	14,75	217,56
	Разом		15	15	15	15	15	15	15	105	0	420,75

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \sum_{j=1}^{N} r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105 \qquad (4.1)$$

де N – число експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 26{,}25 \qquad (4.2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \qquad (4.3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{N} \Delta_i^2 = 420{,}75 \qquad (4.4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)} = \frac{12 \cdot 420{,}75}{7^2(5^3 - 5)} = 1{,}03 > W_k = 0{,}67$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю 4.4

Таблиця 4.4 – Попарне порівняння параметрів

Параметри	Експерти							Кінцева оцінка	Числове значення
	1	2	3	4	5	6	7		
X1 і X2	=	>	=	<	=	<	<	<	0,5
X1 і X3	<	<	<	<	<	<	<	<	0,5
X1 і X4	>	>	>	>	>	>	>	>	1,5
X2 і X3	<	<	<	<	<	<	<	<	0,5
X2 і X4	>	>	>	>	>	>	>	>	1,5
X3 і X4	>	>	>	>	>	>	>	>	1,5

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1{,}5 \text{ при } X_i > X_j \\ 1.0 \text{ при } X_i = X_j \\ 0.5 \text{ при } X_i < X_j \end{cases}$$

З отриманих числових оцінок переваги складемо матрицю $A = \| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступними формулами:

$$K_{\text{вi}} = \frac{b_i}{\sum_{i=1}^{n} b_i}, \text{де } b_i = \sum_{i=1}^{N} a_{ij}.$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$K_{\text{вi}} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, \text{де } b_i' = \sum_{i=1}^{N} a_{ij} b_j.$$

Як видно з таблиці 5.5, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

Таблиця 4.5 – Розрахунок вагомості параметрів

Параметри x_i	Параметри x_j				Перша ітер.		Друга ітер.		Третя ітер	
	X1	X2	X3	X4	b_i	$K_{\text{вi}}$	b_i^1	$K_{\text{вi}}^1$	b_i^2	$K_{\text{вi}}^2$
X1	1,0	0,5	0,5	1,5	3,5	0,219	22,25	0,216	100	0,215
X2	1,5	1,0	0,5	1,5	4,5	0,281	27,25	0,282	124,25	0,283
X3	1,5	1,5	1,0	1,5	5,5	0,344	34,25	0,347	156	0,348
X4	0,5	0,5	0,5	1,0	2,5	0,156	14,25	0,155	64,75	0,154
Всього:					16	1	98	1	445	1

4.3 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій

окремо.

Абсолютні значення параметрів *X2*(об'єм пам'яті для збереження даних) та *X1* (швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра *X3 (*час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 1000 мс або варіанту б) 80мс.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується так (таблиця 4.6):

$$K_K(j) = \sum_{i=1}^{n} K_{вi,j} B_{i,j} \qquad (4.5)$$

де n – кількість параметрів;$K_{вi}$– коефіцієнт вагомості i–го параметра;B_i – оцінка i–го параметра в балах.

Таблиця 4.6 – Розрахунок показників рівня якості варіантів реалізації основних функцій ПП

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1(X1)	А	11000	3,6	0,215	0,774
F2(X2)	А	16	3,4	0,283	0,962
F3(X3,X4)	А	800	2,4	0,348	0,835
	Б	80	1	0,154	0,154

За даними з таблиці 4.6 за формулою

$$K_K = K_{\text{ТУ}}[F_{1k}] + K_{\text{ТУ}}[F_{2k}] + \ldots + K_{\text{ТУ}}[F_{zk}],$$

визначаємо рівень якості кожного з варіантів:

K_{K1} = 0,774 + 0,962 + 0,835 = 2,57

K_{K2} = 0,774 + 0,962 + 0,154 = 1,89

Як видно з розрахунків, кращим є перший варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.4 Економічний аналіз варіантів розробки ПП

Для визначення вартості розробки ПП спочатку проведемо розрахунок трудомісткості.

Всі варіанти включають в себе два окремих завдання:

1. Розробка проекту програмного продукту;

2. Розробка програмної оболонки;

Завдання 1 за ступенем новизни відноситься до групи А, завдання 2 – до групи Б. За складністю алгоритми, які використовуються в завданні 1 належать до групи 1; а в завданні 2 – до групи 3.

Для реалізації завдання 1 використовується довідкова інформація, а завдання 2 використовує інформацію у вигляді даних.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань.

Проведемо розрахунок норм часу на розробку та програмування для кожного з завдань. Загальна трудомісткість обчислюється як

$$T_O = T_P \cdot K_П \cdot K_{СК} \cdot K_M \cdot K_{СТ} \cdot K_{СТ.М},$$

де T_P – трудомісткість розробки ПП;$K_П$ – поправочний коефіцієнт; $K_{СК}$ – коефіцієнт на складність вхідної інформації; K_M – коефіцієнт рівня мови програмування; $K_{СТ}$ – коефіцієнт використання стандартних модулів і прикладних програм; $K_{СТ.М}$ – коефіцієнт стандартного математичного забезпечення.

Для першого завдання, виходячи із норм часу для завдань розрахункового характеру степеню новизни А та групи складності алгоритму 1, трудомісткість дорівнює: T_P =90 людино-днів. Поправочний коефіцієнт, який враховує вид нормативно-довідкової інформації для першого завдання: $K_П$ = 1.7. Поправочний коефіцієнт, який враховує складність контролю вхідної та

вихідної інформації для всіх семи завдань рівний 1: $К_{СК}$ = 1. Оскільки при розробці першого завдання використовуються стандартні модулі, врахуємо це за допомогою коефіцієнта $К_{СТ}$ = 0.8. Тоді, за формулою 5.1, загальна трудомісткість програмування першого завдання дорівнює:

$Т_1$ = 90·1.7·0.8 = 122.4 людино-днів.

Проведемо аналогічні розрахунки для подальших завдань.

Для другого завдання (використовується алгоритм третьої групи складності, степінь новизни Б), тобто $Т_Р$ =27 людино-днів, $К_П$ =0.9,$К_{СК}$ = 1,$К_{СТ}$ =0.8:

$Т_2$ = 27 · 0.9 · 0.8 = 19.44 людино-днів.

Складаємо трудомісткість відповідних завдань для кожного з обраних варіантів реалізації програми, щоб отримати їх трудомісткість:

$Т_I$ = (122.4 + 19.44 + 4.8 + 19.44) · 8 = 1328,64 людино-годин;

$Т_{II}$ = (122.4 + 19.44 + 6.91 + 19.44) · 8 = 1345.52 людино-годин;

Найбільш високу трудомісткість має варіант II.

В розробці беруть участь два програмісти з окладом 7000 грн., один фінансовий аналітик з окладом 9500грн. Визначимо зарплату за годину за формулою:

$$С_Ч = \frac{М}{T_m \cdot t} \text{ грн.},$$

де М – місячний оклад працівників;T_m – кількість робочих днів тиждень;t – кількість робочих годин в день.

$$С_Ч = \frac{7000 + 7000 + 9500}{3 \cdot 21 \cdot 8} = 46{,}62 \text{ грн.}$$

Тоді, розрахуємо заробітну плату за формулою

$$С_{ЗП} = С_Ч \cdot Т_i \cdot К_Д,$$

де $С_Ч$– величина погодинної оплати праці програміста;$Т_i$ – трудомісткість відповідного завдання; $К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 46,62 \cdot 1328.64 \cdot 1.2 = 74340,57$ грн.

II. $С_{ЗП} = 46,62 \cdot 1345.52 \cdot 1.2 = 75285,04$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%:

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 74340,57*0.22 = 16354.93$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 75285,04*0.22 = 16562.71$ грн.

Тепер визначимо витрати на оплату однієї машино-години. ($С_М$)

Так як одна ЕОМ обслуговує одного програміста з окладом 7000 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 7000 \cdot 0,2 = 16800$ грн.

З урахуванням додаткової заробітної плати:

$С_{ЗП} = С_Г \cdot (1 + К_З) = 16800 \cdot (1 + 0.2) = 20160$ грн.

Відрахування на єдиний соціальний внесок:

$С_{ВІД} = С_{ЗП} \cdot 0.22 = 20160 * 0,22 = 4435.20$ грн.

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 10000 грн.

$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 10000 = 2875$ грн.,

де $К_{ТМ}$– коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача; $К_А$– річна норма амортизації; $Ц_{ПР}$– договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 10000 \cdot 0.05 = 575$ грн.,

де $К_Р$– відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою:

$Т_{ЕФ} = (Д_К - Д_В - Д_С - Д_Р) \cdot t_З \cdot К_В = (365 - 104 - 8 - 16) \cdot 8 \cdot 0.9 = 1706.4$ годин,

де $Д_К$ – календарна кількість днів у році; $Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів; $Д_Р$ – кількість днів планових ремонтів устаткування; t –кількість робочих годин в день; $К_В$– коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,9733 \cdot 2,0218 = 523.83$ грн.,

де $N_С$ – середньо-споживча потужність приладу; $К_З$– коефіцієнтом зайнятості приладу; $Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою:

$С_Н = Ц_{ПР} \cdot 0.67 = 10000 \cdot 0,67 = 6700$ грн.

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н$$

$С_{ЕКС} = 20160 + 4435.20 + 2875 + 575 + 523.83 + 6700 = 35269.03$ грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 35269.03 / 1706,4 = 20,67$ грн/час.

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т$$

I. $С_М = 20,67 * 1328,64 = 27462.99$ грн.;

II. $С_М = 20,67 * 1345.52 = 27811.9$ грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67$$

I. $С_Н = 74340,57 * 0,67 = 49808,18$ грн.;

II. $С_Н = 75285,04 * 0,67 = 50440,98$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_М + С_Н$$

I. $С_{ПП} = 74340,57 + 16354.93 + 27462.99 + 49808,18 = 167966.67$ грн.;

II. $С_{ПП} = 75285,04 + 16562.71 + 27811.9 + 50440,98 = 170100.63$ грн.;

4.5 Вибір кращого варіанта ПП техніко-економічного рівня

Розрахуємо коефіцієнт техніко-економічного рівня за формулою:

$$К_{ТЕРj} = К_{Кj} / С_{Фj},$$

$К_{ТЕР1} = 2,57 / 167966.67 = 0,15 \cdot 10^{-4}$;

$К_{ТЕР2}$ = 1,89 / 183561,43 = 0,1·10^{-4};

Як бачимо, найбільш ефективним є перший варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР1}$= 0,15· 10^{-4}.

4.6 Висновки

В даному розділі проведено повний функціонально-вартісний аналіз ПП, який було розроблено в рамках дипломного проекту. Процес аналізу можна умовно розділити на дві частини.

В першій з них проведено дослідження ПП з технічної точки зору: було визначено основні функції ПП та сформовано множину варіантів їх реалізації; на основі обчислених значень параметрів, а також експертних оцінок їх важливості було обчислено коефіцієнт технічного рівня, який і дав змогу визначити оптимальну з технічної точки зору альтернативу реалізації функцій ПП.

Другу частину ФВА присвячено вибору із альтернативних варіантів реалізації найбільш економічно обґрунтованого. Порівняння запропонованих варіантів реалізації в рамках даної частини виконувалось за коефіцієнтом ефективності, для обчислення якого були обчислені такі допоміжні параметри, як трудомісткість, витрати на заробітну плату, накладні витрати.

Після виконання функціонально-вартісного аналізу програмного комплексу що розроблюється, можна зробити висновок, що з альтернатив, що залишились після першого відбору двох варіантів виконання програмного комплексу оптимальним є другий варіант реалізації програмного продукту. У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР1}$= 0,15· 10^{-4}

Цей варіант реалізації програмного продукту має такі параметри:

– мова програмування – Python;

– СКБД MongoDB;

– інтерфейс користувача, створений за технологією Kivy.

Даний варіант виконання програмного комплексу дає користувачу зручний інтерфейс, непоганий функціонал і швидкодію.

ВИСНОВКИ

В ході дипломного проектування було досліджено шляхи парсингу веб-даних у мережі Інтернет, проблеми які виникають. Визначено актуальність задачі, роль програми-парсера. Досліджено інструментарій за допомогою якого можна реалізувати дане завдання, а також проаналізовано варіанти синтаксичного розбору веб-сторінок, імпорту та експорту даних. Глибоко ознайомлено зі способами захисту від парсингу та методами їх обходу.

Перш за все, ми з'ясували проблеми, які виникають під час спроби завантаження веб-сторінки. Розглянули можливості Python для створення веб-парсера. Наступним нашим кроком було обрання правильного рішення для обходу конкретних проблем. Очевидно, що ми написали універсальний клас, в якому правильно описали методи, що дозволяють нам безпомилково завантажити веб-сторінку без втрати даних.

Далі дослідили готові методи та їх алгоритми для аналізу та обробки html документа, створили їх порівняльну характеристику. Прийшли до оптимального рішення – застосувати модуль lxml для парсингу даних, так як методи цієї бібліотеки построєнні на алгоритмах, які розкладають дані html документа в деревоподібну структуру і швидко виконують пошук по заданих параметрах.

Після обробки даних, наступне завдання – правильно зберегти отримані дані і відобразити їх користувачеві на сайті http://dl-cloud.kpi.ua. Отже, так як потрібно зберігати велику кількість даних, а також складних операцій з БД не передбачається, будемо використовувати нереляційні NoSQL бази даних, які простіші в масштабуванні від традиційних реляційних БД.

Зокрема, вибір впав на MongoDB, яка в даний час є найпопулярнішою базою даних NoSQL. MongoDB – доволі швидка об'єктно-документарна база даних. Ідеологічно це якийсь симбіоз між звичною реляційною БД і "ключ-значення" сховищем, і на мій погляд, вельми вдалий. З одного боку, вона дозволяє робити дуже швидкі операції над об'єктом, знаючи його

ідентифікатор, а з іншого, надає потужний інструмент для складних взаємодій. А також для наглядності роботи програми зберегли інформацію у CSV файл.

Крім того описали структуру програми, спроектували такі діаграми: діаграма станів, діаграма потоків даних, діаграма перецедентів, IDEF0 діагарама.

Розробили простий і легко доступний інтерфейс програми для користувача. Було прийнято рішення, що програма повинна бути десктопною і кросплатформеною. Інтерфейс програми розроблений на спеціальному фреймворку для Python – Kivy.

Отже, підсумуємо основні етапи:

- Коректне завантаження веб-сторінки.
- Вилучення з html документа потрібних нам даних.
- Експорт даних у СКБД.
- Перевірка достовірності результатів шляхом збереження даних у файл CSV формату.
- Розробка графічного інтерфейсу для користувача програми.

В останньому розділі роботи проводиться повний функціонально-вартісний аналіз ПП. Після виконання якого можна зробити висновок, що з альтернатив, що залишилися після першого відбору варіантів виконання програмного комплексу оптимальним є перший варіант реалізації.

У нього виявився найкращий показник техніко-економічного рівня якості $К_{ТЕР1}= 0{,}15 \cdot 10^{-4}$. Цей варіант реалізації програмного продукту має такі параметри:

- мова програмування – Python;
- СКБД MongoDB;
- інтерфейс користувача, створений за технологією Kivy.

ПЕРЕЛІК ПОСИЛАНЬ

1. 7 способов защитить сайт от парсинга и как их обойти [Електронний ресурс] – Режим доступу до ресурсу: https://ergonotes.ru/7-sposobov-zashhitit-sayt-ot-parsinga-i-kak-ih-oboyti. - Дата доступу: 06.04.2016.
2. Парсинг: Что? Зачем? Как? [Електронний ресурс] – Режим доступу до ресурсу: http://parsing.valemak.com. – Дата доступу: 24.04.2016.
3. Веб-парсинг на Ruby [Електронний ресурс] – Режим доступу до ресурсу: https://habrahabr.ru/post/252379/. – Дата доступу: 29.04.2016.
4. МЕТОДЫ ПАРСИНГА САЙТОВ [Електронний ресурс] – Режим доступу до ресурсу: http://ponka.vnukov.ru/content/metody-parsinga-saytov. - Дата доступу: 03.05.2016.
5. Grune D. Parsing Techniques - A Practical Guide / D. Grune, C. Jacobs. – Chichester: Originally published by Ellis Horwood, 1990. – 320 с.
6. Aho A. V. The theory of parsing, translation, and compiling / A. V. Aho, J. D. Ullman. – USA: Prentice-Hall, 1972. – 121 с.
7. AngularJS [Електронний ресурс] – Режим доступу до ресурсу: https://angularjs.org/. – Дата доступу: 23.05.2016.
8. Popular Technology. Режим доступу: http://www.populartechnology.net/2010/08/google-scholar-illiteracy-in-pnas.html. – Дата доступу: 23.04.2016.
9. Mitchell R. Web Scraping with Python / Ryan Mitchell. – Boston: O\'Reilly Media, 2015. – 256 с.
10. Пашін В.П. Методичні вказівки до виконання економіко-організаційного розділу дипломних проектів (робіт) бакалаврів і спеціалістів для студентів Інституту прикладного системного аналізу з дисципліни “Економіка та організація виробництва” для студентів спеціальностей 7.080204 - “Соціальна інформатика”, 7.080203 - “Системний аналіз і управління”, 7.080404 – “Інтелектуальні системи прийняття рішень”, 7.080402 - “Інформаційні

технології проектування” / Пашін В.П., Романов В.В., Єгорова Н.В. – К. : НТУУ “КПІ”, 2011. – 118 с.

11. Lxml - XML and HTML with Python [Електронний ресурс] – Режим доступу до ресурсу: http://lxml.de/. – Дата доступу: 28.05.2016.

12. Beautiful Soup 4 Python [Електронний ресурс] – Режим доступу до ресурсу: http://www.pythonforbeginners.com/beautifulsoup/beautifulsoup-4-python. - Дата доступу: 01.06.2016.